M.Sc. Thesis
Master of Digital Media Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Adapting Linguistic Style using Autoencoders

How to *teach* computers to write with style

Valentin Liévin

Kongens Lyngby 2017

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

This thesis reviews the use of neural networks to build a general natural language model and evaluate its application to the task of linguistic style adaptation. We name style adaptation the process of transforming a sentence into another sentence which conveys the same meaning but uses a different linguistic style. This work has been strongly influenced by the algorithm for artistic style proposed in the computer vision field.

Hopefully, the work presented in this thesis will help to create better generative language model with near human performances. We believe that this technology could be a strong asset in the translation industry and could significantly improve current conversational interfaces.

The literature review has motivated the use of the Variational Auto-Encoder to build a continuous representation of the language. During this thesis, we have first conducted an in-depth study of the theory behind this framework and tested it against the case of the MNIST dataset modeling.

Motivated by the sequential natural of the language, we introduced the Recurrent Neural Network architecture and proven that it was compatible with the Variational Auto-Encoder framework by testing it against the modeling of a narrow set of time series.

As a second step, we applied the Variational Auto-Encoder to build a character-level language model. For this purpose, a narrow set of sentences taken from the Large Movie Review Dataset has been used. We reported poor generative performances but good recognition performances has been notably shown by its ability to rephrase unknown sentences.

Thereafter, we justified the choice of the sentiment as a specific case of stylistic feature and we proposed four different approaches for the task of style adaptation. Afterwards, we exploited the good recognition performances of our model to build a simple prototype which can be regarded as a successful case of style adaptation: the prototype has proven to be able to change the sentiment conveyed by simples sentences while addressing the same object.

We hope that the analysis of the Variational Auto-Encoder and its application to the task of linguistic style adaptation will motivate further research in this domain and we are convinced that our methods can be applied to the task of natural language understanding in the near future. Lastly, we propose further research in order to overcome the poor generative performances and apply this model to generative tasks.

Preface

This Master thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring a Master degree in Digital Media Engineering.

Kongens Lyngby, June 29, 2017

Valentin Liévin

Acknowledgements

I would like to thank my supervisor Professor Dr. Ole Winther for his availability and patience in guiding me to complete this thesis project. I am particularly grateful for all the constructive feedback discussions and the warm reception with which the proposition of this project has been received.

Special thanks to Professor Dr. Michael Kai Petersen for the co-supervision of this project, his most helpful feedback and for sharing particularly relevant related works. Furthermore, I would like to thank him for his role as teacher and supervisor during the two years I spent at DTU which have shown to be a most fruitful experience.

I would also like to thank particularly the people working at EasyTranslate for their warm reception and their support. In particular, I would like to thank Peter Ladegaard, Frederik R. Pedersen and Jin Qian for the fruitful discussions we had and the insights about the translation industry I acquired through my collaboration with them.

I am grateful to Catherine Lepez who communicated her passion for the mathematics during my second year of preparatory classes. I couldn't stress more about the quality of her teaching.

Last but not least, I would like to thank my family and friends for their continuous moral support to keep me going during my thesis.

Kongens Lyngby, June 29, 2017

Valentin Liévin

Contents

CHAPTER 1

Introduction

In the computer vision field, deep learning has proven to be efficient at building highly an expressive representation of unstructured data. In particular, neural algorithms are able to capture the artistic style in images and can use this knowledge to create original works. Creating artworks was an ability thought to be only accessible to human intelligence: creative skills are thought to be a human-only ability.

Modeling the natural language is a major challenge in the context of human-computer interactions and is a substantial step toward language interpretation. Nevertheless, progresses observed in the computer vision field do not have an equivalent in the natural language processing field. As a consequence, modeling the language with a focus on the artistic style remains a task open to future work and promises significant scientific outcomes. This is precisely the problem we aim to solve during this project.

In this chapter, we will first define the concept of style with regards to the natural language. Second, we will present specific problems we aim to solve using this technology. Last, we will introduce a plan to tackle this problem.

1.1 Linguistic Style

By definition, the *style* is a particular procedure by which something is done. For example, tennis is defined by its rules but the style of a tennis player is defined by the way he deals with the rules in order to play. According to [LS07], the linguistic style is defined by the choices the author makes to shape a given content within the global rules of the language. A style corresponds to a particular choice of syntax and lexicon. Following this theory, we can divide the speech into two independent components. The first component would be the content, the thought, the idea the speaker wants to transmit. The second component would be the style: the aesthetic choices made by the speaker to shape the speech. As a result, the same content can be expressed with different linguistic styles and a same linguistic style can be used to express different ideas.

Indeed, we can easily observe that different people use a different style of language. Furthermore, while listening to a single speaker, we can observe different ways of using the language depending on the context and the purpose. For example, people will adopt a very particular lexicon and syntax depending on the people they are talking to. In a familiar context, people could simply say *"Give me the water, please."* to ask for water while in a more formal way, one would use a more polite form such as *"May I have water please?"*.

A particular style is often named by its context. For example, we can distinguish the *style of Shakespeare*, the *Victorian Style* or a *formal style*. For this reason, it is impossible to define an exhaustive list of linguistic styles because we can identify as many styles as we find contexts. For instance, [PK99] provides us with a non-exhaustive list of writing styles:

- Words of more than 6 letters
- First-person singular
- Negations
- Articles
- Positive emotions

- Negative emotions
- Causation
- Insight
- Discreptancy
- Tentative
- Social processes
- Past tense
- Present tense
- Inclusive
- Exclusive

To sum up, the speech can be defined by the combination of two components: the content and the style. The style is a choice of lexicon and syntax which shapes the speech depending on the speaker, the context and the purpose.

1.2 Problem

1.2.1 Translation Industry

The translation industry relies heavily on human abilities and remains mostly untouched by the progress of machine learning techniques. In this section, we will introduce the industry and present how it could be impacted by this project.

1.2.1.1 A Growing market

According to the Common Sense Advisory, a translation industry think tank, the size of the translation industry was about $33.5 billion in 2012. Furthermore, in a context of hyper globalization, exchanges of goods and services between countries are now significant. As a consequence, the need to communicate with customers in different countries is on the rise. This is why companies nowadays don't need only to communicate with fellow citizens but with the whole world. For examples, simple things such as instruction manuals have to be translated into every language spoken within a product's market. As the economic exchanges between countries persist, the translation industry is expected to remain in place. indeed, according to the U.S. Bureau of Statistics, the translation industry is expected to grow by 42% between 2010 and 2020[1].

1.2.1.2 Translation Industry and Machine Translation

However, as the automated translation progresses in terms of quality and affordability, traditional translation agencies are potentially endangered. Even if the market is expanding, isn't it likely to see translation agencies loosing their market? Impressive progress has been made in statistical translation and machine learning models are now reaching the human level [Wu+16].

Traditionally, two machine translation methods were used in the industry. First, the rule-based approach with which language was translated using manually coded grammar rules. Second, the statistical which uses very large corpuses and a mathematical model to produce a translation. However, these methods produce only limited outcomes. In the case of Google Translate, the translation was often enough to have an overall understanding of a text but the translation was not accurate enough to carry every detail within the source document. Yet the worst thing is that translated sentences are not

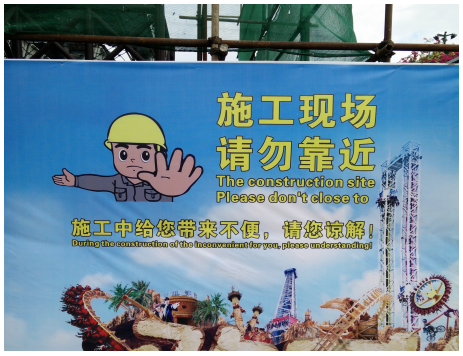


Figure 1.1: On a construction site in Shenzhen. The old machine translation systems were often enough to guess the meaning of a sentence. However, in terms of quality, the results are still far away from a human translation

always grammatically correct as it is shown by the example in the figure 1.1. The automated translation was still far behind the human translation.

In September 2016, [Wu+16] introduced a state of the art machine translation model ready to reach the mass audience. Compared to the previous systems, this model uses neural networks to build a high-level representation of the language. While the previous models were simply matching source and target sentences, the neural models are trained on a large dataset to learn the complex rules of the language by itself. Once it is trained, the model can encode a sentence from a source language into a fixed length thought vector and decodes this vector in the target language using the rules learned by the network.

However, these systems are still limited by the way they are trained: they learn a mapping between two sets of parallel sentences and try to reproduce a translation independently of a broader context. In this way, one sentence in one language corresponds to another in another language. A professional translator, however, will take the context into account in order to adjust the style. Therefore such system works on translating the meaning of the source document but will likely fail in adapting the context and the linguistic style.

1.2.1.3 Human Translation Stays Ahead

In the translation industry, a good translation requires not only to carry the right meaning but also to address the appropriate tonality, sentiment and requires to follow social norms. The translation industry is built on the guarantee that information will be communicated in the right way. Indeed, the translated communication materials are often the only way a company can communicate with its foreign markets. Therefore this work is essential because it will directly affect how the company is perceived and how the company performs abroad. Therefore trust is an important value in the translation industry because the translation agencies are the guarantor of this communication.

As companies place trust in the translation agencies, the translation agencies need to focus on the quality of the translations. This is why the agencies need to ensure that this work will be done by human translators who can deal not only with the language itself but also with the social norms which are specific to a language and a culture. On this point however machine learning models are far behind. Deep learning models are especially criticized for their lack of transparency because these models work like a black box and the quality of the translations are often difficult to assess. Indeed, how to measure if a translation respects specific style? As a consequence, because the automated translation is built on such models, it seems that the traditional translation industry remains ahead.

The lack of quality control and the lack of possibilities to control the quality explain why Machine Learning is not yet a revolution in the translation industry. Existing solutions are translating sentences

[1] https://www.pangeanic.com/knowledge_center/size-of-the-translation-industry/

independently from the context. In this way, we make sure that the translation carries the right meaning. However, using such methods, we can't ensure that the translation will fit the right context, expressed with a particular style.

1.2.2 Conversational Interfaces

The quest for artificial intelligence has focused many efforts and significant progress have been made. However, half a century after the introduction of the Turing Test, personal assistants are still far from passing it.

1.2.2.1 Natural Language as Input and the Rise of the Personal Assistants

Since the beginning of the very first computing devices, inputs are mostly based on touch: keyboard, mouse and touchscreen: all require to use our hands. While this form of interaction is useful in a creative process because the computer becomes an extension of the hand, this form of input is limited for communication and knowledge discovery. For this purpose, the speech is known to be more efficient. First, voice input is faster. Humans can speak 150 words per minutes and can type only 40 words per minute. Second, it is easy to use because it is convenient, hands-free and it doesn't require any training. Instead, voice interfaces use the natural language which makes computing devices accessible to anyone. Third, as the voice interface becomes a universal interface, the size of the devices can decrease. As a consequence, devices such as wearables, IoT objects or even cars become smarter and actionable.

In October 2011, Apple released Siri. Siri was the first personal assistant to reach the mass and was, therefore, the first successful attempt to democratize the voice as a way to control a device. For example, people could ask Siri about their next appointment, they could ask Siri to send a message or they could even ask about general questions such as *"What is the height of the Eiffel Tower?"*

Siri however, isn't the only actor in the market. The tech giants are all developing their own voice assistant: Google Voice, Baidu's voice assistant, Amazon's Alexa, Microsoft's Cortana and Samsung's Bixby. In the meanwhile, the use of these voice assistant is growing. According to KPCB, in the USA, the use of voice assistants has doubled between 2013 and 2015. In China, the voice assistant of the giant Baidu is largely adopted because it is very efficient for the Chinese language which doesn't fit the traditional keyboard input. As a consequence, in 2014, 1 out of 10 Baidu queries came through Speech. Finally, home assistant such as Google Home or Alexa are largely adopted: in 2015, the Amazon Echo (a speaker with Alexa built-in) was the fastest selling speaker with almost 25% of US speaker market.

This large adoption is supported by important technological progress. On the one hand, the voice recognition algorithms have made significant breakthroughs. According to Google CEO Sundar Pichai, the recognition technology has now achieved a 4.9% error rate while the error rate was about 23% in 2013. On the other hand, the systems controlled by voice assistant are more and more complex. For instance, Alexa is now able to control media players using simple voice commands such as "Alexa, play songs by the Beatles". The assistant is also able to get the last news, to recommend movies, to recommend a restaurant and to order a taxi or products to be delivered.

1.2.2.2 Handling Emotions and Simulating Personality

As the technology behind the voice assistant becomes more accessible, smaller actors begin to build their own chatbots, small automated agents made to answers questions or perform tasks through conversation. Chatbots are particularly interesting for companies because it allows them to be more available for their customers. Chatbots are reachable through instant messaging 24/7. However, dealing with customers is a difficult task because they can ask any question and because questions sometimes require some emotional intelligence.

In this direction, [Xu+17] evaluates the use of deep learning to build chatbots for customer service. They observed that 40.5% of the requests were emotional and 59.5% of them were informational. In the case of informational requests, the quality of the answer mostly depends on the information content of the answer. For this reason, the information retrieval method performs well, as the deep learning

method. However, dealing with emotional requests such as *"my flight is canceled."* is difficult for chatbots because the answer isn't a simple information to be retrieved. In this case, a customer expects to have emotional support such as apologies.

Moreover, it is critical for companies to engage with customers on an emotional level because it builds a more intimate relationship. Furthermore, as a company is building a public image, it has to be consistent and find a personal communication style. Most interestingly, in this paper, it has been observed that the model was able to learn writing styles from a brand and transfer them to another. First, this shows the ability of deep learning models to catch a particular linguistic style. Second, it opens the way to create a more advanced brand image: a brand becomes an entity we can talk to on the social media and this entity would have a consistent behavior with a singular personality.

Nowadays, this image is handled by community managers who try not only to answer to informational questions but also to engage emotionally with customers in order to build a brand identity. For example, luxury companies will tend to use a formal lexicon while *young* companies such as Snapchat will tend to use trendy expressions and an informal lexicon. Therefore, building chatbots technology able to target a specific linguistic style would be a solution to build a true persona and this persona would become a new kind of brand identity. Controlling the linguistic style would be an important step in this direction. Indeed, according to [PK99], there is a correlation between an individual's personality and the linguistic style used to write. As a consequence, writing with a consistent and specific linguistic style would produce the illusion of a personality.

Last but not least, building a personality for personal assistant seems to be a path to a stronger user engagement. Siri's personality is a response to the users' needs of entertainment. For example, Siri was built on two different personality traits: Siri is jealous and sassy. For example, to the question *"What is 0 divided by 0?"*, Siri was giving a very abrupt answer*"Imagine that you have zero cookies and you split them evenly among zero friends. How many cookies does each person get? See? It doesn't make sense. And Cookie Monster is sad that there are no cookies, and you are sad that you have no friends."*. This particular answer generated considerable commentary and public interest which has led to a growing user base. We can easily imagine that an informational answer such as *"you can't divide a number by zero"* would not have helped the assistant to become popular. Thus, creating a personality-based personal assistant fosters engagement.

1.2.3 The Lack of Control over linguistic Style as a common Obstacle

In the case of the translation industry and the construction of advanced chatbots, we identified a common problem. In both cases, progress is limited by the ability of Machine Learning systems to control their linguistic style because they are modeled and train to carry the right meaning only. In the case of the translation industry, the inability of the automated translation methods to fine tune the translation to fit a particular context is the main obstacle. Controlling the writing style could be a way to increase dramatically the quality of automated translations. In the case of the conversational interfaces, the current systems could be greatly improved by providing a way to write in a particular style. In this way, we open the possibility to build pseudo personalities which could be used to build a powerful brand identity and engage users on an emotional level. Thus, controlling the stylistic features in the context of text generation is a worthy challenge.

1.2.4 A New Hope: Style, Machine Learning and Computer Vision

In the computer vision field, [GEB15] has shown the ability of neural networks to transfer the style from an artwork to another image. Using such model, it is now possible to turn a simple picture into an artwork which mimics the artistic style of a famous painter 1.3. Looking from a different perspective, this transformation preserves the information contained in the scene: the picture represents the same objects or persons which are still recognizable. However, the way these elements are pictured is modified: objects are represented with a different artistic style. This work is particularly interesting because it

shows the ability of neural networks to build a high-level representation of images in which different components for the contents and the style can be identified and modified independently.

1.3 Objective

The last advanced in the Machine Learning field have shown the ability of neural networks to interpret images in a very high-level fashion. Using such representation, it is now possible to deal with high-level concepts such as the artistic style. Using such method, it was possible to transfer the style from an artwork to another. Following this line of thought, we would like to apply such concept to the linguistic field. In this way, we could transform sentences such as they convey the same information while changing linguistic features such as the semantic field, level of politeness or emotion. For example, we would like to transform the sentence *"Give me the water, please"* into a more polite form as *"May I have water please?"*. In this example, the main message stays the same: a request for water. However, the second sentence is much more appropriate. Using this technology, we hope to improve the translation industry by providing an on-demand translation with a language style perfectly adjusted to the client needs. We also hope to improve chatbots technologies by providing a way for the bots to express themselves in a unique manner with aim of imitating personality.

To sum up, the aim of this project is to develop a natural language model and investigate the use of this model for the task of style adaptation.

1.4 Related Work

This project is directly inspired by the recent progress in the field of computer vision and neural-style algorithm proposed by [GEB15]. However, we will present projects related to our problem which are most likely to be useful in order to define a technical solution to the linguistic style adaptation problem.

1.4.1 Neural Style: a multi-modal representation of images

[GEB15] has demonstrated the ability of very deep convolutional neural networks to acquire a high-level representation of images with separate components for the content and the style. This paper uses the VGG Network, a very deep convolutional neural network used for image recognition purpose.

The VGG network transforms the image into a set of high-level features using the different layers sequentially. Interestingly, some features are very explicit. For example, two pictures of dogs with a totally different arrangement of pixels will share a common feature corresponding to the concept "dog". Because these features represent explicit content, we call them *content features*.

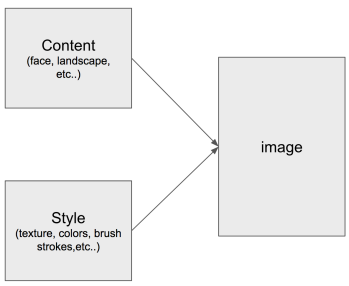


Figure 1.2: The high level representation of an image can be separated into two independent components: style and content

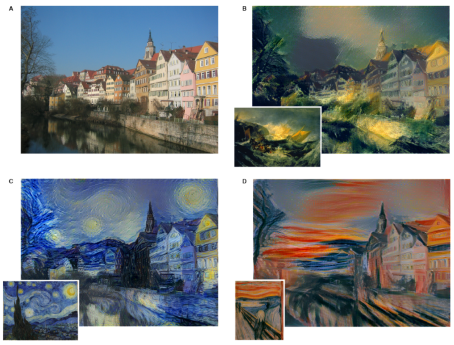


Figure 1.3: Deep learning models are able to change the style of an image without modifying its content. The original image (top right) is transformed into new artworks (other images) that mimic the artistic style of different painters. While the image represents the same thing (the houses are still recognizable), the artistic style is changed. Reproduced from [GEB15]

On the other hand, other features were dedicated to capture the texture of an image. Textures features were capturing local structures and colors. Thus, these stylistic features are a *style representation* of an image: the subject of an image is defined by the content features but the local arrangements are independently controlled by the stylistic features.

Using such representation, it is now possible to transfer the artistic style of an image by replacing the style component of this image with the style component from another image. A few examples are shown in the figure 1.3. This technique met a consequent success and has found practical applications. For instance, the mobile application Prisma use this technique to transform any picture into an original artwork (figure 1.4).

In this project, we aim to apply this concept to the natural language field. Thus, the first step towards this direction will be to build a model with a multi-modal representation of the language with distinct style and content representations.

1.4.2 Neural Photo Editing: modifying pictures using a latent representation

[Bro+16] shows how a high-level representation of images can be used in a creative creative context. In this paper, a latent representation of a set of images is built using a highly expressive generative model (VAE-GAN). Using the latent representation of the images, the user is able to interact with high-level concepts. For instance, using a dataset of portraits, the user is able to modify the hairstyle or to add a beard to the subject while keeping a photo-realistic result (figure 1.5). This is yet another approach to interact with high-level concepts using Deep Learning. Furthermore, this paper is particularly interesting because it shows that specific features can be adjusted independently from the other.

Figure 1.4: Artworks created using the Prisma app. Prisma uses the neural representation of images to transfer the artistic style from famous paintings to any picture

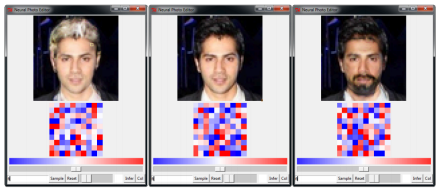


Figure 1.5: Neural Photo editing allows the user to transform a picture by interacting with high-level features. For instance, in this picture, the user modifies the hairstyle by modifying the latent space.

Input	Output
but you're not listening to me.	but you do not hear me .
Gregory, on my word, we will not be humiliated, like carrying coal.	regory , we 'll not carry coals .
but he got the promotion.	he is the friend .
i can hit quickly, if i'm motivated.	i strike , i am moved .
Did you just give us the finger, sir?	have you leave the thumb , sir ?
You don't know what you're doing!	you do not what you know you .
have you killed Tybalt?	hast thou slain tybalt ?
Why, Romeo, are you crazy?	why , art thou mad , mad ?

Figure 1.6: A sequence2sequence model is able to translate modern English into Shakespeare's English: an interesting example of linguistic style transfer

1.4.3 Controlling the Linguistic Style

Adapting linguistic feature is not a new idea and some attempts have already been performed. [Xu+12] proposes a model to translate modern English into Shakespeare's. A few samples are available in the figure 1.6. In this case, the problem was treated as a translation problem using sequence2sequence model introduced by [Cho+14]. The sequence2sequence model encodes a source sentence into a thought vector and decodes it into the target language. This allows translating one sentence into another language.

This approach produces good results. However such method requires having access to a large parallel corpus from one style to another. This is why it would be difficult to apply the method to other stylistic features.

[SHB16] uses yet another approach to adapt the linguistic style. This paper proposes a neural translation model which can translate English into German using a formal or informal style. Some samples are available in the figure 1.7. In this case, the challenge was to build a multi-modal translation model such as one source sentence can be translated into two different versions: a formal sentence and a familiar one. For this purpose, side constraints are used. Sentences were labeled as formal and informal and the model was trained taken this parameter as input. After training, it was possible to adjust the politeness by modulating this parameter.

This approach is particularly interesting because it doesn't depend on a parallel corpus with sentences available in two different styles. Instead, side constraints were built using labeling. Furthermore, such system can be trained using semi-supervised techniques which reduce further the need for labeled specimen.

1.4.4 A Continuous and High-level Representation for the Natural Language

The two papers introduced in the previous sections show how deep learning can be used to handle photographic data in a high-level fashion. [Bow+15] builds a generative model for natural language featuring a highly expressive latent representation of the language.

In this paper, a Variational Auto-Encoder is used to model a very large corpus of text. Interestingly, the obtained latent representation is continuous with regard to the meaning and generated sentences are grammatically correct. As a consequence, points sampled from a path between two sentences in the latent space will be decoded into grammatically correct sentences with an intermediate meaning from the starting sentence to the end as shown in the table 1.1.

source	Give me the telephone!
reference	Gib mir das Telefon! [T]
none	Gib mir das Telefon! [T]
polite	Geben Sie mir das Telefon! [V]
informal	Gib mir das Telefon! [T]
source	Are you kidding?
reference	Das ist doch ein Witz! [N] (this is a joke!)
none	Machst du Witze? [T]
polite	Machen Sie Witze? [V]
informal	Machst du Witze? [T]
source	You foolish boy.
reference	Du dummer Junge. [T]
none	Du dummer Junge. [T]
polite	Du dummer Junge. [T]
informal	Du dummer Junge. [T]

Figure 1.7: Side constraints can be used with a neural translation model to adapt the degree of politeness

These results are most interesting because it is a first step towards building a highly expressive model for natural language that would allow us to interact with it in a high-level fashion such as it has been observed in the field of computer vision. In the case of the sequence2sequence model introduced in the previous example, side constraints had to be used to generate variations of the same sentence. In the case of the Variational Auto-Encoder, the continuity of the latent space ensures that these variations can be found within this latent representation.

1.4.5 Unsupervised Models for Sentiment Analysis

In[RJS17], a Recurrent Neural Network is trained in an unsupervised fashion to predict the next character in Amazon reviews. After training, reviews are labeled with their corresponding sentiments: negative review or positive. Surprisingly, one neuron is highly characteristic of the reviews sentiment (figure 1.8). It is yet another example that shows that neural network can model high-level concepts such as a particular stylistic feature.

Thereafter, the model is used to generate reviews by itself. By providing an initial vector, the recurrent neural network produces a sentence iteratively. In this context, we can constrain the network to generate positive or negative reviews. With the same initial configuration but with different values for the sentiment neuron, the network can generate reviews with a similar object and different sentiment. For instance, using positive sentiment neuron we obtain:

i want to talk to you .
i want to be with you .
i do n't want to be with you .
i do n't want to be with you .
she did n't want to be with him .
he was silent for a long moment .
he was silent for a moment .
it was quiet for a moment .
it was dark and cold .
there was a pause .
it was my turn .

Table 1.1: Interpolations between two sentences

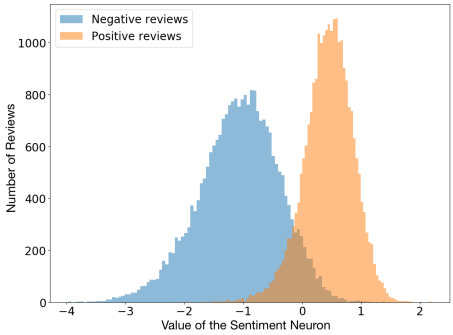


Figure 1.8: After having been trained in an unsupervised way to predict the next character, a neuron in the model can be used to classify positive and negative reviews

I couldn't figure out the shape at first but it definitely does what it's meant to do. It's a great product and I recommend it highly.

On the opposite, with a negative sentiment neuron, the network produces:

I couldn't figure out how to use the product. It did not work.At least there was no quality control; this tablet does not work. I would have given it zero stars, but that was not an option.

Most interestingly, this example[2] shows that unsupervised models can learn high-level features such as the sentiment. Nevertheless, even if such model can be used for the generation of random snippets of text, it seems impracticable to exploit it to adapt linguistic style. Indeed, this task would require knowing the corresponding initial vector corresponding to the target snippet of text.

1.5 Approach

In the field of computer vision, neural networks have shown to be able to acquire a multi-modal representation of the data. In this representation, we can identify very expressive features corresponding to high-level concepts such as the subject of the picture: *"the picture shows a cat or a dog"* or the artistic style: *"this is a photo-realistic picture, this is a painting from Van Gogh"*.

In the field of natural language, unsupervised neural models have shown to be surprisingly good at modeling language and high-level features such as sentiment in an unsupervised fashion. Furthermore, the Variational Auto-Encoder has proven to be able to construct a high-level and continuous representation of the language which is very relevant in the context of style modeling.

Consequently, we propose to use the Variational Auto-encoder framework as described by [Bow+15] to build a natural language model. Using this representation of the language. First, we will investigate the latent space for the expression of stylistic features such as the positivity or negativity. Secondly, we will use this representation in order to adapt linguistic style.

According to [PK99], the use of positive or negative emotions are the expression of a particular linguistic style and datasets for sentiment analysis are largely available. Therefore, the task of adapting the sentiment is a good candidate to investigate the use of the Variational Auto-Encoder in the context of style adaptation.

To accelerate the development of this project, we will work iteratively. Each step will consist in attacking a simple problem and evaluating the proposed solutions. In this way, we will be able to exploit the outcomes of the previous steps with confidence.

[2] This example has been reproduced from the original paper.

More precisely, we will first develop a vanilla Variational Auto-Encoder for non-sequential data structures such as simple pictures. Second, we will build a toy task using simulated time series with the aim of building a Variational Auto-Encoder designed for sequential data. Third, we will adapt this framework to the modeling of natural language. Lastly, we will adapt our general language model to the task of style adaptation.

To sum up, we will deal with this problem in an iterative fashion. Small problems will be defined and be attacked sequentially. The outcome of each step will be used to tackle the next problem. The plan of this project can be summarized as follows:

1. Introducing and experimenting the Variational Auto-Encoder Framework
2. Presenting and experimenting an extension of the Variational Auto-Encoder Framework designed for sequential data: the Variational Recurrent Auto-Encoder
3. Applying the Variational Recurrent Auto-Encoder to the task of natural language modeling and assessing the performances
4. Investigating the use of our model for the task of linguistic style adaptation with a focus on the case of the sentiment

CHAPTER 2

Variational Auto-Encoder

In this chapter we introduce the Variational Auto-Encoder framework introduced by [KW13]. First of all, we will address the basics theory on the top of which the VAE is built. Second, we will deal with the theory behind the framework itself. Finally we will apply the VAE framework to basic examples such as the well-known MNIST dataset and discuss the results.

2.1 Theoretical background

The Variational Auto-Encoder is built on the top of neural network and relies heavily on the Bayesian statistics. In order to study this framework in the best conditions, we must first construct a solid knowledge basis of the underlying concepts.

2.1.1 Information Theory and Entropy

A digital image of the same screen size can require a different storage size depending on the content and the compression algorithm. For example, for .jpeg images, an image composed of black pixels only will be lighter than a picture of a cat. On the other hand, telling France's history will take more time than explaining a cooking recipe. In both cases, we could measure the quantity of information using the disk space in the first case and the size of the book in the second case. However, these different metrics are not comparable.

As a response to this problem, [Nyq24] and then [Har28] introduced a measure of the quantity of information:

$$H = \log S^n$$

where S represents the number of possible symbols (for example 0 or 1 for binary signals) and n the length of the transmission. This formula provides us with an unbiased way to measure the capacity of a media or a transmission device.

However, this measure of information is limited because the optimal encoding depends also on our knowledge of the world. Indeed, frequent pieces of information should use fewer bits to be encoded since they occur more frequently. This observation justifies the need for a measure which is not only able to measure the quantity of information as an absolute value (without context) but as a measure the unexpected: a measure of surprise. For example, during the summer period, knowing that the weather is sunny in the South of France does not provide much information because it occurs very frequently. Therefore this information has little value because it doesn't teach us anything new. However, if the same day we hear that it will snow in the South of France, it is unexpected because this event is not common. Thus, the quantity of information contained in a message depends on the frequency of occurrence of the same message.

Following this line, Shanon has introduced H as the entropy of information:

$$H = \sum_i p_i \log \frac{1}{p_i} = -\sum_i p_i \log p_i$$

where p_i is the probability of each symbol to be communicated. This formula provides us the mean to measure the performance of an encoding: if a message contains redundant pieces of information.

Then this message doesn't carry much information compared to its size. Therefore we expect a low entropy for this message. However, if the message is optimal regarding the entropy, then every piece of the message is useful (and unexpected): the entropy of the message is high.

2.1.2 Bayesian probabilities, Bayes' Theorem and Bayesian Inference

2.1.2.1 Bayesian probabilities

The frequentist statistics depend on the fixed observation of the frequency of an event given a large number of trials. Therefore, frequentists assume that their dataset is large enough to be a true representation of the world. On the other hand, The Bayesian statistics make a reasonable hypothesis about the state of the world given their actual knowledge. Then, the hypothesis is updated in the light of new evidence.

2.1.2.2 Bayesian's Theorem

The Bayes' Theorem gives us a relation between the probability of two events:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

2.1.2.3 Bayesian Inference

In order to make inferences, we use the Baye's theorem as followed:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

where H represents a hypothesis, E represents an evidence, P(H) represents the probability of observing H the prior knowledge, P(E) stands for the probability of the event observed with the experiment E, P(H|E) is the conditional probability of the event based on our knowledge (hypothesis) given the

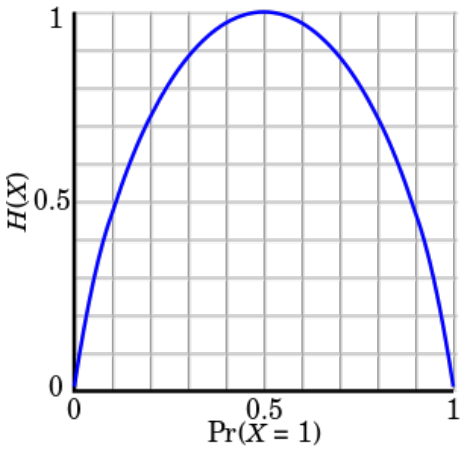


Figure 2.1: Plot of the Binary Entropy function. The binary entropy function is a special case of the Entropy and provides a good illustration of this key concept of the Information Theory. Taking X as a tossing coin event, if the coin is unbalanced, then $p(X = 0) \sim 1$ or $p(X = 1) \sim 1$ then the entropy is very low: the outcome of the experiment is unlikely to be surprising because it is likely to be the same at every trial. However, if the coin is balanced, then $p(X = 1) = p(X = 0) = 0.5$ and the outcome is difficult to predict. Therefore each result is *surprising*: the entropy is very high. Credits: Wikipedia

evidence E and P(E|H) represents the probability of observing E given H).
In other words, Bayesian inference allows us to update a probability for a hypothesis given a observable evidence: it allows us to update our knowledge given an observation. Therefore Bayesian inference is extremely important in for machine learning given that it enables us to update our model with observation (i.e. data).

2.1.3 Latent Variable Model

A latent variable is defined as a variable affecting an observation without being directly observable. For instance, in Economics, latent variables can be defined by the quality of life, morale, happiness or conservative. These variables have an observable effect on the economy but can't be directly observed. Latent variable models are statistical models which treat observable variables as a posterior of an unobservable variable and define a relationship between them.

Latent variables are also very relevant for generative models. While the posterior distribution is of very high dimensionality represent low-level concepts (for example, the value of a pixel in an image), the prior distribution is often of low dimensionality and represents higher level concepts such as the content of the image itself or the artistic style in the case of a painting.

2.1.4 Kullback-Leibler Divergence

The KL Divergence is a measure between two probability distributions. The KL divergence from Q to P is written $D_{KL}(P||Q)$ and is not symmetric, therefore $D_{KL}(P||Q) \neq D_{KL}(P||Q)$. Nevertheless, the KL Divergence has vocation to be a metric and satisfies $D_{KL} \geq 0$.

Given $H(P, Q)$ the cross entropy of P and Q, the KL Divergence is defined to be:

$$\begin{aligned} D_{KL}(P||Q) &= \sum_i P(i) \log \frac{P(i)}{Q(i)} \\ &= -\sum_i P(i) \log Q(i) + \sum_i P(i) \log P(i) \\ &= H(P, Q) - H(P) \end{aligned} \tag{2.1}$$

The KL Divergence is a really interesting term in the domain of Bayesian Inference. Indeed, this term can be interpreted as a measure of the information which is gained when our model changes his belief from his prior knowledge to the posterior knowledge.

In a more pragmatic way, this term can be simply used to measure a distance between two distributions. Subsequently, this allows us to evaluate the quality of an estimator. Given a probability density function P, if we try to approximate this distribution with another function Q then the KL Divergence

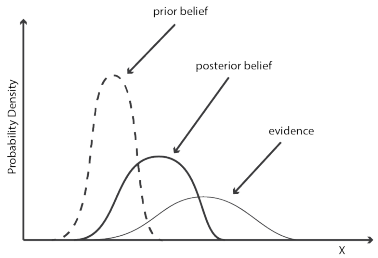


Figure 2.2: Bayesian inference allows us to update our belief given an experiment

can be used to measure the distance between the two distribution. Thus measuring the quality of the estimator Q.

2.1.5 Neural Networks

In 1943, [MP43], introduced the *thresholded logic unit*: a simple model designed to mimic the way a neuron works. This artificial neuron takes one or more inputs and sums them to produce an output which depends on a threshold. The output of a neuron is defined as:

$$y_k = \varphi\left(\sum_{j=0}^{m} w_{kj} x_j\right)$$

Most significantly, a group of such units could be used for pattern recognition: each neuron could fire or not depending on its set of inputs and its activation level. This is what we call a neural network: a group of artificial neurons tied together. Neural Networks can be used for various purposes. for instance, based on this theory, in 1958, [Ros58] build the perceptron: a machine able to recognize simple visual patterns based on artificial neurons. The machine was taking an array of 400 photocells connected to neurons with potentiometers acting as weights. The perceptron is a direct application of McCulloch's thresholded logic unit. Given f a threshold operation, the perceptron was defined as:

$$o = f\left(\sum_{j=0}^{m} i_j x_j\right) \tag{2.2}$$

However, the perceptron has been quickly rejected by the scientific community because of its incapacity to model simple non-linear functions such as the XOR function.

After a long pause, the resurgence of the neural networks happened in the 80's with the democratization of the back-propagation algorithm which made possible the training of multi-layered models which are a solution to model more complex functions. In the meantime, the universal approximation theorem has shown that multi-layer perceptrons were not only able to approximate the XOR function but theoretically *every* function.

Nonetheless, neural networks weren't very popular and methods such as SVMs were the favorite. it is only after 2012 that Neural Networks regained in popularity with the work of [Hin+12] which introduced a state-of-the-art model for speech recognition based on deep neural networks. This regain of popularity can be mainly explained by two factors:

- the availability of large and labeled dataset: deep learning algorithms are very greedy
- the affordability of GPU computing which enable to train larger models

Today, deep learning algorithms are achieving state-of-the-art results in multiple domains such as image recognition, image captioning and music generation.

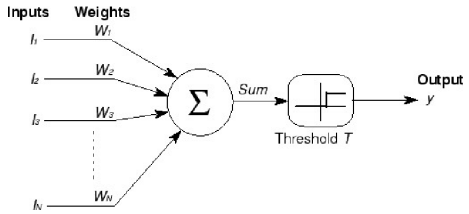


Figure 2.3: The threshold Logic Unit was the first kind of artificial neuron. The artificial neuron sums up input to produce a binary output. Credits: ResearchGate

2.1.5.1 Multi-layer Perceptron

A *multi-layer perceptron* (MLP) or *deep feedforward network* is a perceptron with multiple layers of artificial neurons.

The goal of MLP is to approximate a function f^* in order to learn a mapping $y = f^*(x)$. In the case of the perceptron, given ϕ a given activation function, we defined the model as a single layer network such as:

$$y = \phi(Wx + b)$$

In the case of MLP, we simply compose layers together to form a single model. Each intermediate layer, called a hidden layer, takes the outputs of the previous layer as inputs. Each hidden layer can be considered as a different level of abstraction. The use of multiple layers explains why we name them *Deep* Neural Network.

Formally, we define a MLP composed of N layers as following:

$$\begin{cases} y_0 = x \\ y_{i+1} = \phi(W_i y_i + b_i), i \in [0, N-1] \\ y = y_N \end{cases}$$

2.1.5.2 Gradient descent

Gradient descent is an iterative optimization algorithm. The core idea is to consider the cost function F as a surface parametrized by the parameters θ of the model. Because neural networks are continuous and differentiable at every point, we can compute the *slope* (i.e. gradients) at the current point and move down the surface (i.e. update θ) towards a local minimum.

Given a dataset X, a model $f(X, \theta)$ and F the cost function, γ an hyper-parameter (learning rate), we can evaluate iteratively the parameter θ with regards to the cost function F:

$$\forall n, \theta_{n+1} = \theta_n + \gamma F(\theta_n))$$

2.1.5.3 Back-propagation

The universal approximation theorem states than a MLP can approximate any function. However, this theorem doesn't touch upon the learnability of the parameters. Unfortunately, training multi-layered neural network is a difficult task because it requires differentiating the cost function with regard to every parameter and parameters can be very numerous. For each parameter θ_i, we would need to compute:

$$\frac{\partial F}{\partial \theta_i} = \frac{F(\theta + \epsilon e_i) - F(\theta)}{\epsilon}$$

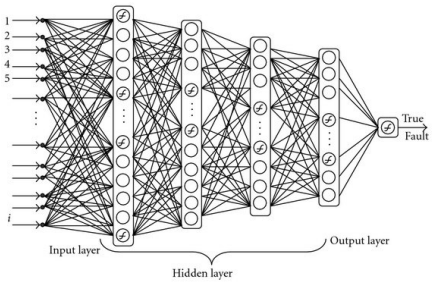


Figure 2.4: A Multi-Layer Perceptron is composed of multiple layers. According to the universal approximation theorem, a MLP can approximate any function. Credits: Quora

This operation requires one forward passes through the network for each parameter which is prohibitively expensive to perform. Fortunately, deep neural networks formed of multiple layers, each of them corresponding to an independent function. Therefore deep neural networks are compositions of functions:

$$f = f^{(1)}of^{(2)}of^{(3)}...of^{(n)}$$

The back-propagation algorithm uses the chain rule to compute the gradients using only one forward pass and one backward pass.

First, the forward pass takes the input vector and pass it through the network in order to evaluate the error as defined by the cost function.

Second, we will use the current value of the error to compute the gradients of the last layer with regard to the error. Fortunately, we can re-use the gradients of the last layer to evaluate the next layers thanks to the chain rule. For instance, let be $y = g(x)$ and $z = f(g(x)) = f(y)$. Then, in the case of a scalar, the chain rule states that:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

The neural network can be easily *unrolled* using the chain rule following a computational graph as the one shown in the figure 2.5. It allows us to evaluate iteratively the partial derivates using the chain rule.

Third, we can use the evaluation of the partial derivates in order to update the parameters as described by the gradient descent algorithm.

2.1.5.4 Mini-batch and Stochastic Gradient Descent

The Gradient Descent requires to evaluate the error on the entire dataset which can be prohibitively expensive. For instance, the WMT dataset used for machine translation consists in ~ 5 million pairs of sequences. We would need to evaluate the partial cost function for each of the pairs of sentences at each iteration.

The Stochastic Gradient Descent algorithm (SGD) is a response to this problem. Instead of computing the true gradients based on the entire dataset, SGD proposes to compute an approximation of the gradient based on a few samples (mini-batch). Thus, the SGD algorithm may not lead to the right direction at each step. However, in average it is likely to lead towards a local minimum.

2.1.5.5 Supervised and Unsupervised Learning

In machine learning, we say that an algorithm learns in a supervised fashioned if the model is trained using labeled data. For example, a classifier that learns to classify hand-written digits will be trained with both a list of arrays of pixels: the image itself and with a list of labels: the actual digit (”0”, ”1”, etc..).

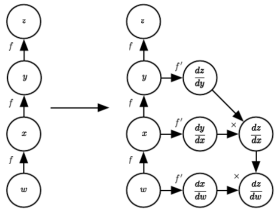


Figure 2.5: The backpropagation algorithm is based on the iterative evaluation of the partial derivates which can be easily unrolled following computational graph. Credits: DeepLearningBook.org

On the contrary, an algorithm that learns without any other clue than the data points themselves will be labeled as unsupervised. Generally, these algorithms acquire an inner representation of the structure of the data by identifying similar items. They are often used for clustering purpose.

However, it is important to notice that even models trained in an unsupervised way are used in a supervised context. For example, the Principal Component Analysis or the Negative Matrix Factorization algorithms can be used to build a representation of a dataset with a lower dimensionality. This representation allows us to find clusters corresponding to the different classes in the model space. The model space, corresponding to a transformation of the pixels space has been obtained in an unsupervised fashion. Nonetheless, the use of these results to classify new digits is a supervised algorithm because it requires knowing which digit corresponds to which cluster.

In other words, unsupervised algorithms have the ability to find underlying structure in a set of objects but the rapprochement of these structures with human concepts is nothing else than a supervised process because concepts are human interpretation.

2.1.5.6 Softmax and Classification Tasks

Neural Networks are commonly used for classification purpose. For a binary classification, we can use the sigmoid function to map the output into the range $[0, 1]$, the output would represent the probability for an item to be of the class A or $\overline{A}$.

Similarly, we can classify objects between multiple class using one output for each class. However, we need to normalize the outputs to sum the probabilities up to 1.

In machine learning, the $Softmax$ function or normalized exponential function is used for normalization purpose in the context of classification of mutually exclusive classes. Given an output vector of dimension K, the softmax function is defined as:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, , K.$$

Additionally, the exponential nature of the $Softmax$ function is particularly interesting in the context of neural networks because such models are often trained using the log likelihood as an objective function.

2.2 The Variational Auto-Encoder Framework

We will now present the underlying problems encountered when building generative models. Then we will introduce the Variational Bayesian method and its application to the Variational Auto-Encoder. Lastly, we will compare the framework with its deterministic equivalent.

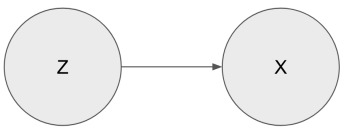


Figure 2.6: In the context of generative models, we consider that a continous variable X is generated from a random process involving a latent variable Z

2.2.1 Generative Models

Generating data such as music, images or speech has been a longstanding problem. For example, generating pictures of human faces requires first to know that a face features two eyes, a nose, lips, eyebrows, hairs and potentially facial hairs. The next step will be to know how to organize them all together in a meaningful way. Last, the model must be able to render a realistic lighting which depends on the size, intensity, and position of the light source. We are therefore facing a very complicated problem, especially knowing that the machine only deals with a matrix of pixels.

We represent the ensemble of all the possible images of faces by a continuous variable X. $p(X)$ represents the probability for an image to be an image of a face, $p(X)$ is a thin manifold in the image space, observable through a set of samples of x. However, X is often of a very high dimensionality and samples of X are very sparse. Consequently, it is difficult to find the probability density function $p(X)$ which could be used generate samples x.

Instead, in the context of a generative model, we assume that the variable X is generated from a random process involving a hidden random variable z: a latent variable of a lower dimensionality. For instance, this latent variable could represent a set of features such as the length of the nose, the color of the eyes, etc.. Finally, we define the generative process as a two-step mechanism:

- Firt, samples $z^{(i)}$ are generated from a prior distribution $p_{\theta^*}(z)$.
- Second, the samples $x^{(i)}$ are generated from a given conditional distribution $p_{\theta^*}(x|z)$.

it is important to notice that this process depends on an unknown parameter θ^*. Because this parameter is unknown, we aim to find the maximum of the marginal likelihood: $\theta = \arg\max_{\theta} \log p(x \mid \theta)$. Indeed, we want to maximize $\log p(X)$ because we want to make sure that every point from the latent space leads to a valid specimen (i.e. $p(X) = 1$).

However, computing the marginal likelihood requires to evaluate $p(X)$ and, unfortunately, $p(X)$ is intractable. Indeed, the equation 2.3 shows that $p(X)$ depends on the unknown posterior probability $p(x|z)$.

$$\begin{aligned} p(x) &= \int p(z, x) dz \\ &= \int p(z) p_\theta(x|z) dz \end{aligned} \tag{2.3}$$

Solutions exist but come with their own disadvantages. Traditionally, two methods are used to approximate the posterior distribution:

- Using Markov Chain Monte Carlo (MCMC) methods which involve an expensive iterative process for each data point. As a consequence, this method doesn't scale to large datasets.
- Making strong simplifying assumptions on the posterior distribution. For example, we could assume that $p_\theta(x|z)$ would be a diagonal Gaussian. However, this assumption might not reflect the complexity of the real world.

The Variational Auto-Encoder (VAE) proposes a novel solution to this problem. Instead of using expensive sampling methods (MCMC) or using strong assumptions, [KW13] and [JMW14] implemented the Variational Inference method on the top of standard optimization algorithms such as SGD. The variational Auto-Encoder solves two problems: scalability and the need to make strong assumptions.

2.2.2 Variational Bayesian Method

Instead of using an expensive sampling method to approximate the true posterior distribution, the Variational Bayesian method consists in using optimization in order to approximate the intractable true posterior $p_\theta(z|x)$. Fo this purpose, we approximate $p_\theta(z|x)$ using an auxiliary function $q_\phi(z|x)$ called the recognition model[1]. Using the recognition model, we can now propose a new expression of the marginal likelihood as shown by the equation 2.4. However, in the equation 2.4, the term $D_{KL}(q(z^{(i)}|x)||p_\theta(z|x^{(i)}))$ is still problematic because its dependency on $p_\theta(z|x^{(i)})$. Consequently, computing the likelihood is still out of range.

$$\begin{aligned}
\forall i, \log p(x^{(i)}) &= \sum_z q(z|x^{(i)}) \log p(x^{(i)}) \\
&= \sum_z q_\phi(z|x^{(i)}) \left[\frac{q_\phi(z|x^{(i)}) p_\theta(x^{(i)}, z)}{p_\theta(z|x^{(i)}) q_\phi(z|x^{(i)})} \right] \\
&= \sum_z q_\phi(z|x^{(i)}) \log \frac{q_\phi(z^{(i)}|x)}{p_\theta(z|x^{(i)})} + \sum_z q(z|x^{(i)}) \log p_\theta(x, z) - \sum_z q_\phi(z|x^{(i)}) \log q_\phi(z|x^{(i)}) \\
&= D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})) + E_{q_\phi(z|x^{(i)})}[-\log q_\phi(z|x^{(i)}) + \log p_\theta(x^{(i)}, z)] \\
&= D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z|x^{(i)})) + \mathcal{L}(\theta, \phi; x^{(i)})
\end{aligned} \tag{2.4}$$

Nevertheless, in the context of a generative model, we are not required to evaluate the term $\log p(X)$, we need to maximize it. Fortunately, $D_{KL}(q(z^{(i)}|x)||p_\theta(z|x^{(i)})) >= 0$ because the Kullback-Leibler Divergence is non-negative. Therefore, we can define a Variational Lower Bound of $\log p(X^{(i)})$ such as:

$$\log p(X) \geq \mathcal{L}(\theta, \phi; x^{(i)}) = E_{q(z^{(i)}|x)}[-\log q(z^{(i)}|x) + \log p_\theta(x, z)]$$

We can easily observe that summing up over the mini-batch keeps the inequality:

$$\log p(X) = \sum_{i=1}^{N} p(x^{(i)}) \geq \sum_{i=1}^{N} \mathcal{L}(\theta, \phi; \theta^{(i)})$$

Last, the Variational Lower Bound can be expressed in a different and more convenient form:

$$\begin{aligned}
\mathcal{L}(\theta, \phi; x^{(i)}) &= E_{q_\phi(z|x^{(i)})}[-\log q_\phi(z|x^{(i)}) + \log p_\theta(x^{(i)}, z)] \\
&= \sum_z q_\phi(z|x^{(i)}) \log q_\phi(z|x^{(i)}) - \sum_z q_\phi(z|x^{(i)}) \log p_\theta(z)) + \sum_z q_\phi(z|x^{(i)}) \log p_\theta(x^{(i)}|z) \\
&= -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) + E_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)]
\end{aligned} \tag{2.5}$$

In the equation 2.8, every term of $\mathcal{L}(\theta, \phi; x^{(i)})$ can be easily expressed and computed. The left term $\mathcal{L}_l = -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))$ will be called the *latent loss* or *regularization loss*. This term measures the quality of the estimator $q_\phi(z|x^{(i)})$ with respect to the target distribution $p_\theta(z)$ (a choice which depends on the use case). The second term $\mathcal{L}_r = E_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)]$ measures the ability of the model to generate valid specimen. We will name it *reconstruction loss*.

2.2.3 Variational Auto-Encoder

The Variational Auto-Encoder is a direct application of the Variational Inference using Neural Networks. As defined in the previous section, we want to model an unobservable variable Z to control the generation

[1] [KW13] proposes to call the recognition network $p_\theta(z|x)$ a probabilistic encoder and the generative network $p_\theta(z|x)$ an probabilistic decoder since this model encodes a datapoint x into a code z and decodes it into a datapoint x.

of an observable random variable X. In this context, the Variational Bayesian method identifies three key elements:

1. the generative model $p_\theta(x|z)$
2. the recognition model $q_\phi(z|x)$ which approximates $p_\theta(z|x)$
3. the lower bound of the marginal likelihood $\mathcal{L}(\theta, \phi; \theta^{(i)})$

The latent loss $\mathcal{L}_l = -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))$ depends on $p_\theta(z)$ which is an unknown distribution. We choose to use an isotropic Gaussian to shape the distribution of z:

$$p_\theta(z) = \mathcal{N}(z; 0, I) \tag{2.6}$$

Furthermore, the posterior distribution $p_\phi(z|x)$ is still free of taking any form. We will assume that the true posterior distribution is a circular Gaussian with parameters $\sigma^{(i)}$ and $\mu^{(i)}$ as shown in the equation 2.7. As a result, for each datapoint, its representation in the latent space will be a diagonal Gaussian controlled by two parameters which will be controlled by the recognition model.

$$q_\phi(z^{(i)}) = \mathcal{N}(z, \mu^{(i)}, \sigma^{(i)} I) \tag{2.7}$$

These two choices may seem oversimplifying. Nonetheless, no assumption is made on the posterior $p_\theta(x|z)$ and we can find a function mapping the latent variable Z to the observation variable X, we *simply* need a powerful approximator for $p_\theta(x|z)$. The ability of the model to generate complex data structure only depends on the ability of the generative model to model complex mappings. Thus, it is important to stress that the generative model needs to be a particularly strong approximator.

In the context of this framework, the recognition and generative models will be implemented using Neural Networks and the Variational Lower Bound will be used as an objective function. At training time, the model is implemented as a feed-forward neural network. This is interesting because it allows us to use optimization algorithms such as Stochastic Gradient Descent. For this purpose, we need to evaluate the loss function over a mini-batch and propagate the gradients through the network. The backpropagation algorithm requires each operation in the graph to be differentiable. However, as shown in the figure 2.7, reconstructing a data point $x^{(i)}$ requires to sample $z \sim \mathcal{N}(z, \mu^{(i)}, \sigma^{(i)} I)$. Unfortunately, the sampling operation is not differentiable. As a consequence, the gradients can't be propagated through the sampling operation.

In response to this problem, [KW13] and [JMW14][2] propose to reparametrize the sampling operation. Instead of defining z as a random variable drawn from $\mathcal{N}(z, \mu^{(i)}, \sigma^{(i)} I)$, z is expressed as the result of a deterministic function which takes an independent random variable as input:

[2] [KW13] and [JMW14] are cross-referencing each other

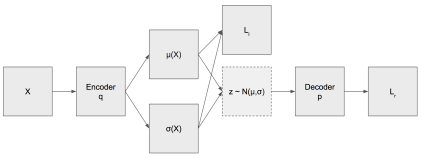


Figure 2.7: At training time, the model is implemented as a feed-forward neural network. Using backpropagation, the gradients are propagated through the network in order to update the parameters ϕ and θ. However, this requires every operation to be differentiable. Reconstructing a datapoint $x^{(i)}$ requires to sample $z \sim \mathcal{N}(z, \mu^{(i)}, \sigma^{(i)} I)$ and the sampling operation is not differentiable

$$z = g_\phi(\epsilon, x) \text{ with } \epsilon \sim p(\epsilon)$$

More specifically, as $z \sim \mathcal{N}(z, \mu^{(i)}, \sigma^{(i)} I)$, we have:

$$z = \mu(x) + \epsilon * \sigma(x) \text{ with } \epsilon \sim \mathcal{N}(0, 1)$$

The flow chart updated with the implementation of the reparametrization trick is shown by the figure 2.8. As we can observe, we can now find a path from the reconstruction loss to the recognition model which doesn't involve non-differentiable operations such as sampling.

To sum up, we have built a generative model which can theoretically model complex data structures because no assumption has been made on the posterior distribution $p_\theta(x|z)$. Assumptions are only made about the prior distribution and the ability of the model to generate complex objects depends on the efficiency of the generative model. Fortunately, this model is implemented using neural networks which have shown to be able to learn complex non-linear dependencies. Last, the Variational Auto-Encoder is scalable because it is built on the top of Neural Networks and is fully trainable using SGD. For this reason, it doesn't involve expensive computationally expensive methods relying on looping over every sample (MCMC).

2.2.4 Stochastic versus Deterministic Models

Stochastic models differ from deterministic models in the use or non-use of random variables. More specifically, the Variational Auto-Encoder is a stochastic model because of the sampling operation $z \sim q_\phi(z|x)$. By contrast, a vanilla Auto-Encoder which simply transforms data points into a higher-level representation using layers of a lower dimensionality and without sampling operation is a deterministic model. Deterministic Auto-Encoders can be applied to extract high-level features by lowering the dimensionality of the data. Thus, the question arises: why should we use the Variational Auto-Encoder which seems to be overly complex and is known for its difficulty to be trained?

This choice depends on the problem to be tackled. In the context of this project, first, we aim to build a generative model such as samples taken from the latent space are decoded into valid data points. Indeed, in the context of the style adaptation, we will need to generate sentences with a different style. Thus, a generative model is required. Second, we aim to build smooth latent representation which would allow us to transform one data point to another in a continuous fashion. Regarding the style adaptation, a smooth representation would mean that similar items are likely to be placed close to another. Indeed, we assume that sentences with a different style but a similar meaning share a great similarity. As a consequence, a smooth space would ensure that sentences with similar meanings are

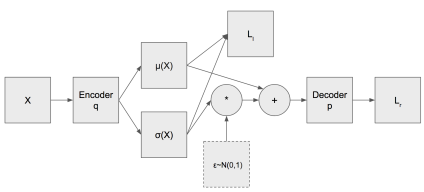


Figure 2.8: The reparametrization tricks allows the gradients to be propagated from the evaluation of the error ($\mathcal{L}_r$) and $\mathcal{L}_l$)) to the parameters parameters θ and ϕ of the generative and recognition models p and q

represented in the same neighborhood of the latent space. To sum up, this problem requires a *dense* and smooth representation[3].

Density The stochastic nature of the Variational Auto-Encoder and the constraints on the distribution $p(z)$ are the guarantee of the creation of a *dense* and *smooth* latent representation. Regarding the *density* property, the key difference between the two methods lies in the differences in the definition of the problem itself. Indeed, a deterministic model will encode an input vector into another low-dimension vector without applying constraints on the overall distribution of the latent representations. By contrast, the Variational Auto-Encoder deals explicitly with the distribution of the latent representations. More precisely, this distribution is assumed to be $\mathcal{N}(0, 1)$. On the other hand, the objective of the model is to maximize the log likelihood $\log p(x) = \int_z p(x, z)dz$ which pushes every point $z \sim \mathcal{N}(0, 1)$ to be valid. In other words, the latent space is filling up with valid latent representations of x, hence the *density* property is obtained.

Smooth Representation Concerning the smoothness of the latent representation, we can argue that this property is a consequence of the adversarial effects of the approximation of the posterior distribution $p_\theta(z|x)$ assumed to be diagonal Gaussian and the constraints imposed to $p(z)$. First, we have seen that data points are represented as distributions of $q_\phi(z|x) = \mathcal{N}(\sigma_x, \mu_x)$. By using such representation, we ensure that every data point is represented with a non-null *width* σ.[4] As a consequence, the model is pushed to store many data points with a non-null width representation into a space of a finite capacity. This process forces the model to optimize the available space to fit the whole dataset. In practice, this is done by placing similar items close to another because similar items share some properties which means they can partially share the same code. Thus, the need to optimize the available space leads to the construction of a smooth representation.

By contrast, none of these properties is expected to be true in the context of a vanilla Auto-Encoder. The latent representation of such model is unbounded and not *dense* wich prevents us from using such model in a generative context.

2.3 Implementing the Variational Auto-Encoder

In the previous chapters, we introduced the Variational Auto-Encoder framework with a theoretical point of view. We will now focus on the implementation of the model using Tensorflow. First, we will present the general implementation of the model. Second, we will showcase the model using a simple example: modeling pictures of handwritten digits.

2.3.1 Architecture

At training time, the Variational Auto-Encoder is an extension of the Auto-Encoder framework and is trained as a feed-forward neural network.

- First, the encoding or recognition model encodes a datapoint $x^{(i)}$ into its latent representation $q_\phi(z^{(i)}) = \mathcal{N}(\mu^{(i)}, \sigma^{(i)}I)$.
- Second, the stochastic layer samples the posterior distribution: $z \sim \mathcal{N}(\mu^{(i)}, \sigma^{(i)}I)$.
- Third the decoder function $p_\theta(x^{(i)}|z)$ decodes the samples z into the original datapoint x.

[3] By using the term *dense*, we intend to say that the latent space is filled up with valid codes z: codes which are decoded into correct data-points x.

[4] The use of the random process $z \sim \mathcal{N}(\sigma_x, \mu_x)$ is simply a way to *implement* the distribution $q_\phi(z|x)$.

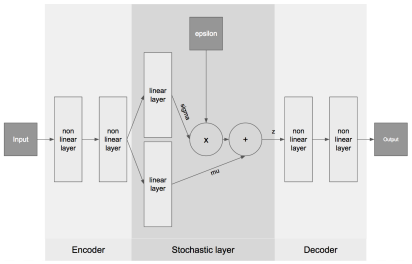


Figure 2.9: The architecture of the VAE. The Encoding and Decoding layers are implemented using 2 non-linear layers. The stochastic layer is implemented using the parametrization trick to ensure the possibility for back propagation. The latent distribution is modeled using mixtures of Gaussian. Therefore the distribution is represented using two parameters: μ and σ

Regarding the recognition and generative model (q and p), any kind of neural network can be used if they can be trained as a feed-forward neural network: Multi-Layer Perceptrons (MLP), Recurrent Neural Networks (RNN) or Convolutional Neural Networks (CNN).

For simple problems such as the modeling of the MNIST dataset introduced in ([LC10]), we will chose to use simple MLP layers to model the encoding network. Moreover, in the section 2.2.2, we have stated that $p_\theta(z|x)$ is assumed to be diagonal Gaussian and controlled by two parameters μ and σ. Thus, we can express these parameters as following:

$$\mu(x) = LINEAR(MLP(x)) \text{ and } \sigma(x) = LINEAR(MLP(x))$$

The stochastic layer represents the latent representation of the data-points. When using the model as a generative model, we sample $z \sim p_\theta(z)$. Moreover, we have to make sure that the model is trainable using backpropagation. As described in section 2.2.2, we implement the reparametrization trick such as:

$$z = \mu(x) + \epsilon * \sigma(x) \text{ and } \epsilon \sim \mathcal{N}(0, 1)$$

Finally, the generation model will be modeled as a MLP with a final linear layer to adjust the output to the right dimension:

$$x_{reconstruction} = LINEAR(MLP(z))$$

The complete architecture is represented in the figure 2.9.

2.3.2 Training

The model is trained using Stochastic Gradient Descent (SGD). For each mini-batch, the gradients are evaluated using the objective function. Then, the gradients are propagated backward through the graph and the parameters (θ, ϕ) are updated.

In the section 2.2.2, we have defined the objective function as the Variational Lower bound:

$$\mathcal{L} = \mathcal{L}_l + \mathcal{L}_r$$

$$\text{with } \mathcal{L}_l = -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z))$$

$$\text{and } \mathcal{L}_r = E_{q_\phi(z|x^{(i)})}[\log p_\theta(x^{(i)}|z)].$$

Fortunately, TensorFlow computes gradients automatically. For this purpose, we need to provide the framework with an explicit expression of the objective function.

2.3.2.1 Expression of the Latent Loss $\mathcal{L}_l$

The latent loss measures the quality of the estimator q with respect to the assumed posterior distribution $p_\theta(z|x) = \mathcal{N}(0,1)$.

If J is the dimension of z, we have:

$$\begin{aligned}
\mathcal{L}_l(\theta, \phi; x^{(i)}) &= -D_{KL}(q_\phi(z|x^{(i)})||p_\theta(z)) \\
&= \int q_\phi(z|x^{(i)}) \log p_\theta(z) dz - \int q_\phi(z|x^{(i)}) \log q_\phi(z|x^{(i)}) dz \\
&= \int \mathcal{N}(\mu, \sigma^2) \log \mathcal{N}(0,1) dz - \int \mathcal{N}(\mu, \sigma^2) \log \mathcal{N}(\mu, \sigma^2 I) dz \\
&= -\frac{1}{2} \sum_{j=1}^{J} (\mu_j^2 + \sigma_j^2) + \frac{1}{2} \sum_{j=1}^{J} (1 + \log \sigma_j^2) \\
&= \frac{1}{2} \sum_{j=1}^{J} (1 + \log \sigma_j^2 - \mu_j^2 - \sigma_j^2)
\end{aligned} \tag{2.8}$$

2.3.2.2 Expression of the reconstruction Loss $\mathcal{L}_r$

The reconstruction loss measures the quality of the generated points x which are a reconstruction of the input data points. Indeed, $\mathcal{L}_r$ is the expected value of x given z. As z is the encoding of the input x, we have:

$$\mathcal{L}_r = \|x - x_{reconstruction}\|$$

The expression of the quality of the reconstruction depends on the problem. For example, for a regression problem, we will tend to use the Root-Mean-Square-Error (RMSE) while we will tend to use the cross entropy for classification problems.

2.3.3 Experiment on The MNIST Dataset

In this section, we will implement the Variational Auto-Encoder using Tensorflow and we will showcase its use on a simple toy task

2.3.3.1 The MNIST Dataset

The MNIST dataset ([LC10]) is a dataset of pictures of hand-written digits. The database contains 70 000 examples (60 000 for the training set and 10 000 for the testing set). Each image is a binary array of 28x28 pixels. Each image is centered, normalized and comes with its label: an integer in [0,9]. The MNIST dataset is traditionally used in the machine learning community as a benchmarking dataset for the comparison and development of various machine learning models.

2.3.3.2 Setting up the Problem

In this section, we aim to build a generative model for hand-written digits. we aim to find a continuous latent variable z which produces valid elements x using the generative model $p_\theta(x|z)$. Each element x is a binary array of size 28x28. The model will be fed with 60 000 examples without any other clue such as the corresponding labels or any other feature.

2.3.3.3 Implementation

The code is available in the appendix A as a Jupyter Notebook. The recognition and generative models are implemented using a two-layered multilayer perceptron with respectively 400 units for the first layers and 200 units for the second layers. Only two dimensions are used for the latent space for the sake of the ease of visualization.

Regarding the reconstruction loss $\mathcal{L}_r$, we implement it as the binary cross entropy[5]:

$$\mathcal{L}_r(\theta, \phi; \theta^{(i)}) = H(x^{(i)}, x_r^{(i)}) = -\sum_{j=1}^{N}(x_j^{(i)} \log x_{r_j}^{(i)} + (1 - x_j^{(i)}) \log(1 - x_{r_j}^{(i)}))$$

2.3.3.4 Results

Reconstruction The figure 2.10 shows the reconstruction of a few samples taken from the dataset. The reconstructions are not perfect and the results could be greatly improved using more complex neural networks for the recognition and generative models. Nonetheless, these results are sufficient to show the usability of the Variational Auto-Encoder in a generative context. Additionally, we observe that latent representations of the inputs are distributed over $\mathcal{N}(0, 1)$: the sample is centered around $(0, 0)$ and most of the points are located within a radius of 1.

Latent Space The figure 2.11 provides a visualization of the latent space. Images are decoded from different configurations of z. From this visualization, we can notice the following points:

- Every digit is represented in the latent space which seems to indicates that the inner representation of the model covers the whole hand-written digit manifold in the image space.
- Every point in the latent space corresponds to a correct reconstruction (the reconstructed output represents a hand-written digit). Thus, we have successfully built a generative model for hand-written digits.
- The latent space is continuous: small variations of z come with a small variation of the generated samples with regard to high-level features such as the writing style.

The figure 2.12 provides another example of continuous variations in the latent space. Each image decoded from interpolations are valid hand-written digits.

Stylistic features in the Latent Space The aim of this project is to model the language as two different components: content and style. In the context of the hand-written digits, the same concept can be applied. The content a picture of a hand-written digit is its meaning: the digit itself. The style of the hand-written digit is the way the digit is drawn: it can be the overall orientation or a particular curvature. First, in the figure 2.11, we observe that the latent representation of the model is able to catch the content of each class of the dataset because each digit is represented in the latent space. Second, we observe that the latent space is not only constituted of one example for each class but is constituted of a multitude of different specimens and the different specimen from a same class are drawn with a different style.

Most interestingly, the different items from the different classes share common stylistic features. For instance, the x-axis of the latent space seems to have an influence on the orientation of the digits. The ones displayed in the center are straight while the ones on the left or on the right are leaning in a different direction. We observe the same tendency for the digits 5, 6 and 0 although this effect is less visible.

The fact that stylistic features depend on the same properties in the latent space is a first hint in the direction of using such model to modify the style of high-level objects such as the hand-written digits.

[5]The expected predict output must be in [0,1]. Thus, we apply the sigmoid function to the output of the decoder

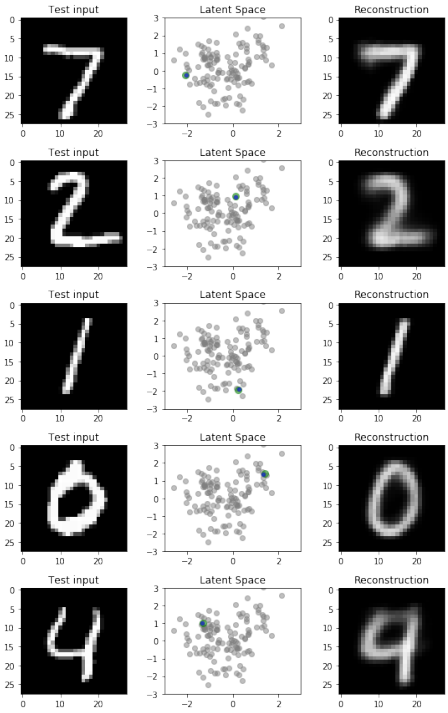


Figure 2.10: Reconstruction of handwritten digits by a simple VAE model. The left column shows the true inputs x, the middle column shows the latent representation of the elements of the mini-batch $q_\phi(z|x)$ and the current element. The right column represents the products of the generative model $p_\theta(x|z)$ (i.e. reconstructions)

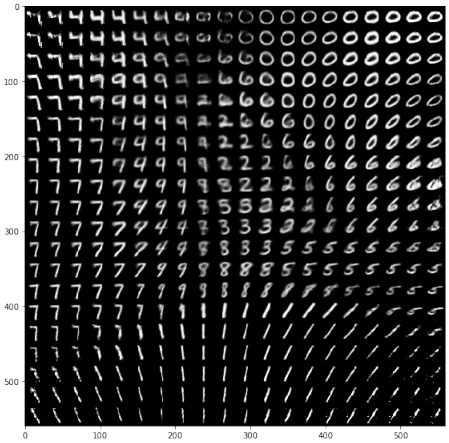


Figure 2.11: Vizualization of the latent space learnt by the Variational Auto-Encoder. This figure is an array representing images x generated with $p_\theta(x|z)$ and with samples of z. Each position (x, y) in the figure space corresponds to the relative combination of $z = (z_1, z_2) \in [-3, 3]^2$. As we can observe, the model has learnt a continous representation of the hand-written digits space

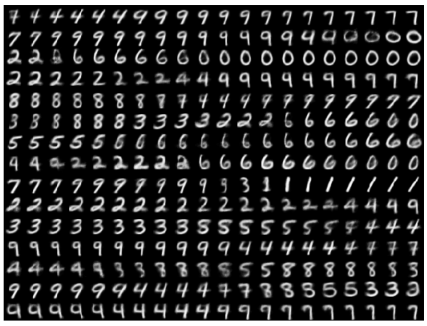


Figure 2.12: Interpolations between two random points in the latent space. Each row corresponds to a different serie.

2.4 Conclusion

First, we introduced the basics of information theory, the Bayesian probabilities, neural networks and the Variational Bayesian inference. Second, we proposed the Variation Auto-Encoder framework as an implementation of the Variational Bayesian method. Last, we tested the Variation Auto-Encoder against the task of modeling the MNIST dataset. We observed that the model can create a complex latent representation of the initial dataset and that continuous variations in this space produce continuous variations on the hand-written digits manifold. Furthermore, we observed that our model can model different styles of hand writing and that the stylistic features are correlated to orientations in the latent space.

The Variational Auto-Encoder used in this chapter is basic and can hardly be applied to more complex uses. The next step will be to develop a more complex model with the aim of modeling more complex objects. More specifically, we will focus on developing a model suited to sequential data.

CHAPTER 3

Variational Recurrent Auto-Encoder

In the previous chapter, we introduced the Variational Auto-Encoder framework and tested it against a simple example: generating pictures of handwritten digits. Unfortunately, this framework doesn't scale to more complicated data structures such as time series or natural language. For instance, we count $(28 * 28)^2 = 614,656$ possible combinations for a binary image of the size of the ones used in the MNIST dataset while for sentences of only 10 words and vocabulary limited to 10,000 words we count $10000^2 = 100,000,000$ different combinations. A problem that seems too complex to be tackled using simple multi-layered neural networks.

However, both in the case of the time series and the sentences, we can observe dependencies between time steps. As a consequence, a term within a sequence can often be predicted using the previous terms. In this section, we will introduce an extension of the VAE framework adapted to sequential data: The Variational Recurrent Autoencoder.

3.1 Recurrent Neural Networks

A common use case of a neural network is the approximation of a posterior distribution $p(y|x)$ using a function $f : x \rightarrow f(x)$ with $f(x) = p(y|x)$ (figure 3.1).

In the case of sequential data such as time series, sentences or videos, it becomes difficult to find the correct sequence Y given an input X because of the very high dimensionality of the data. On the other hand, the structure of such data is very helpful: in a sequence y it is much easier to predict the element y_t knowing the previous elements y_{t-1}, y_{t-2},etc.

For example, let imagine that we have a single neural network which is able to predict the next word in a sentence iteratively. Then for each word x_t in the sentence, there would be a function f such as $x_{t+1} = f(x_t)$. As a result, we could model the function f using an arbitrary neural network and generate sentences easily as shown by the figure 3.2.

This technique may apply to a very small dataset, nonetheless, it is impossible to find such function f for larger corpus because a same word can be followed by different words. In order to model a real-world corpus of sentences, we need to build a model which is able to transmit a history of all the previous elements. In this way, at each step, the neural network would be able to make a prediction using not only the previous element but the three or more past elements. A simple way to implement this manually would be to feed more inputs to the function f. Using the example of the sentence, we can feed one dimension for the t-th word and one dimension for the (t-1)-th word (figure 3.3):

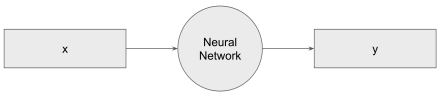


Figure 3.1: Simple neural networks learn to map inputs x with outputs y

Figure 3.2: A single neural network could be used to generate sequential data by predicting every next element using the previous one

- For the first step the neural network will predict "like", store it in the first dimension and place "trains" in the second dimension
- For the second step the neural network will predict "trains" using the two dimensions "I" and "like". Then it will store "trains" in the second dimension for further predictions.

Most interesingly, this is in total accordance with the core concept because we define a function f which predicts X_{t+1} given X_t with $X_t = \begin{pmatrix} x_t \\ x_{t-1} \end{pmatrix}$. We have: $\begin{pmatrix} x_t \\ x_{t-1} \end{pmatrix} = f \begin{pmatrix} x_{t-1} \\ x_{t-2} \end{pmatrix}$.

3.1.1 Formal Definition

We introduced the core concept underlying the Recurrent Neural Network (RNN) and their ability to transmit information. Now we can propose a formal definition of this architecture. An RNN cell, which is used recursively for each time step, takes two inputs: the previous hidden state s_{t-1} which represents the memory of the network, updated at each step and the current input x_t which corresponds to the t-th element of a sequence and can be left void. Then the function learns a transformation f, invariant in time, such as:

$$\begin{pmatrix} s_t \\ o_t \end{pmatrix} = f \begin{pmatrix} s_{t-1} \\ x_t \end{pmatrix}$$

At each step, the RNN cell yields the updated hidden state s_t and the output value y_t. The cell is represented in the figure 3.4. Any kind of network can be used as an RNN cell: multi-layer perceptron, convolutional neural network, LSTM, GRU, etc..

Moreover, this architecture is particularly interesting because the gradients can be propagated in the network following the connections within the unstacked network which make the RNN fully compatible with standard optimization methods and with other Deep Learning frameworks.

3.1.2 RNN as an encoder

3.1.2.1 Theory

We have seen that the Recurrent Neural Networks are able to transmit information by updating a hidden state vector s_t. In fact, this hidden state acts as a summary which retains the information present in the previous steps. Therefore, using an adequate objective function, if we select the last hidden state we could obtain a summary of the whole input sequence, as described in [Cho+14]. In other terms, an RNN can be used to create a fixed-length representation of a sequence. Moreover, as the hidden state is

Figure 3.3: Recurrent Neural Networks can use more dimensions to store previous elements which are used to predict next elements

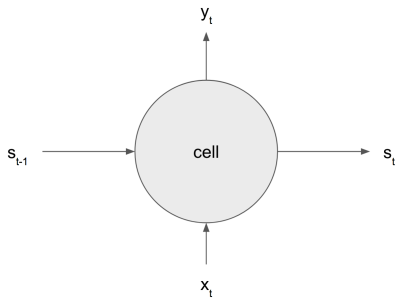


Figure 3.4: A RNN cell takes a previous hidden state s_{t-1} and an optional input x_t as inputs and emit an output y_t and the upadted hidden state s_t

a fixed size vector, its information capacity is limited. Thus, such neural network may have to perform a form of compression to transmit the information which presence is fostered by the objective function.

3.1.2.2 Example

In order to demonstrate the ability of RNN to summarize information, we would like to build a simple example. In this example, we consider a sinusoid signal generated with $t \rightarrow sin(\pi t)$ and a time vector t sampled constantly. We would like to predict the value at the step $T + u$ given T first points (fig. 3.6). This task not only requires knowing the value of the previous point but also the variation between the points in the previous steps.

In order to perform this prediction, we encode the input sequences using a simple RNN with a simple non-linear layer as a cell. given the inputs x_t, W and b the weights and biases of the non linear layer, we construct the hidden state such as:

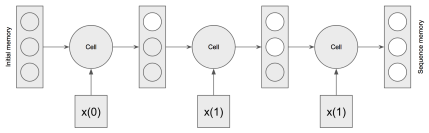


Figure 3.5: A RNN is used to encode a sequence into a fixed length vector.

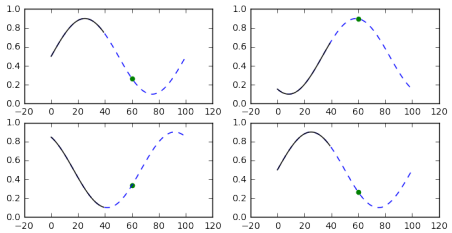


Figure 3.6: An RNN is trained to predict the value of the point T+u given a sequence of T observations sampled from a sinusoid. This would demonstrate the use of an RNN as an encoder since this task requires to known about the previous steps to infer the value of the point at the step $T + u$. This graph shows the training data: the full line is the sequence of observations x and the green point is the label y

$$s_{t+1} = f\begin{pmatrix} s_t \\ x_t \end{pmatrix} = \tanh(\begin{pmatrix} s_t \\ x_t \end{pmatrix} W + b)$$

We implement this model using Tensorflow as follows:

Listing 3.1: Implementation of the Encoder RNN

```
    1
1   def static_rnn(inputs):
2       # initial memory
3       initial_memory = tf.zeros([batch_size, state_size], dtype=dtype)
4       memory = initial_memory
5       rnn_inputs = tf.unstack(inputs, signal_size, axis=1)
6       for x in rnn_inputs:
7           # group inputs (previous memory s_t-1 + current input x_t)
8           cell_inputs = tf.concat( [memory, x], 1)
9           # applying the operation f performed by the cell
10          memory = tf.nn.tanh( tf.add( tf.matmul( cell_inputs , W), b ) )
11      return memory
```

In order evaluate the next value after u steps, we use a fully connected layer to "interpret" the hidden state and train the model using the RMSE as the objective function. After training, the model successfully predicts the value of the step $T+u$. The full code is available on my Github [1]. This example demonstrates the ability of RNN to encode the information contained in a whole input sequence (figure 3.7).

3.1.3 RNN as a decoder

3.1.3.1 Theory

The RNN can also be used as a generative model. Given an initial state vector, the network can produce results iteratively. For example, given the information representing a sentence, an RNN could produce each word one after another. In this case, however, it is important to notice that the dimension of the hidden state can be different than the dimension of the data. Therefore an additional layer has to be added to the neural network to project the cell state into the data space. A simple non-linear layer can be used for this purpose.

[1] https://github.com/vlievin/Tensorflow-tutorials

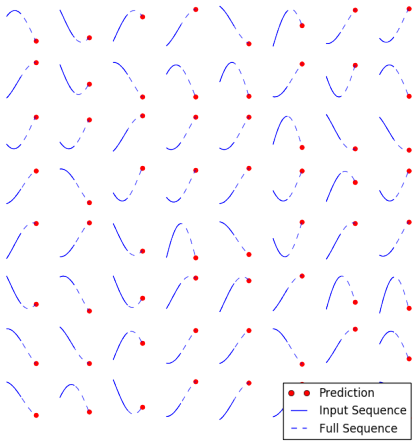


Figure 3.7: Using more dimensions, we can transmit previous elements to the next steps

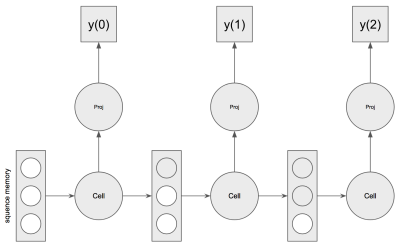


Figure 3.8: A decoder network takes an input as initial state vector and produce each element of a sequence iteratively. As the dimension of the hidden state and the dimension of the data don't necessarly match, we use a projection layer

3.1.3.2 Example

To demonstrate the use of such model, we use the same example as described in the section 3.1.2.2 and try to predict not only one point but a whole serie of points.

Again, we use a non-linear layer as a cell with W and b the weights and biases of the RNN cell. In this case, we use a linear layer as projection layer with W_{proj} and b_{proj} the weights and biases. We can define:

$$\begin{pmatrix} s_{t+1} \\ y_t \end{pmatrix} = \begin{pmatrix} \tanh(s_t W + b) \\ s_{t+1} W_{proj} + b_{proj} \end{pmatrix}$$

The decoder is implemented as follows using Tensorflow:

Listing 3.2: Implementation of the Decoder RNN

```
   1
1  def decoder(inputs_encoding):
2      W = tf.Variable(tf.random_uniform([state_size, state_size], 0, 1), dtype=dtype)
3      b = tf.Variable(tf.zeros( [state_size] ), dtype=dtype)
4      hidden_state = inputs_encoding
5      outputs = []
6      for _ in range(signal_size):
7          # update the hidden state
8          hidden_state = tf.nn.tanh( tf.add( tf.matmul( hidden_state , W), b ) )
9          # project the output
10         output = tf.add( tf.matmul( hidden_state , W_proj), b_proj )
11         outputs.append(output)
12     outputs = tf.stack(outputs,1)
13     return outputs
```

Using the RMSE as objective function, we train the model and obtain a very good reconstruction (fig. 3.9). The code is available on my Github [2].

3.2 The Variational Recurrent Auto-Encoder

3.2.1 Introduction

The Variational Recurrent Autoencoder (VRAE) is basically a Variational Autoencoder (VAE) with the recognition and generative models implemented as Recurrent Neural Networks (RNN).

In [Fv14], the original paper, the model was applied to the generation of short pieces of music and was trained on video games musics such as the theme of Super Mario. Using the VAE framework applied to images (section 2.3.3), we obtained a latent representation of the images. In this latent space, the

[2] https://github.com/vlievin/Tensorflow-tutorials

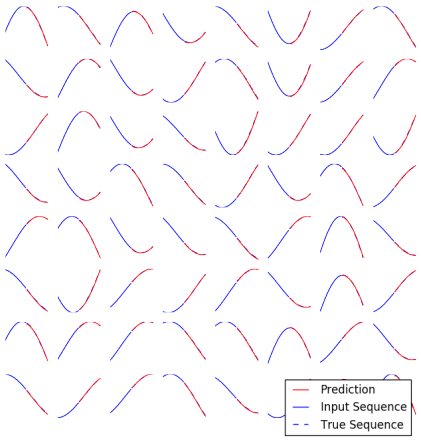


Figure 3.9: The decoder RNN is able to interpret the encoding vector produced by the encoder network to predict the next T points

same digits were grouped together. Similarly, the VRAE model is able to encode a set of music samples into a low dimension latent space. From this latent space, we are able to generate short music by drawing variable from the latent space. Similarly to the case of the VAE, we observe that continuous variations in the latent space come with continuous variations in the generated music samples. This property also shows that the model is able to generate brand-new samples since we can generate an infinite number of new samples because of the continuous property of the model.

In this chapter, we are going to implement the VRAE framework with some improvements. Then we apply the model to a simple task: the generation of time series where the series are points sampled from sinusoids with variable phases and frequencies.

3.2.2 The Model

The model is an extension of the VAE presented in the previous chapter. The overall architecture of the model is similar except for the implementation of the encoder and decoder networks which are made using RNNs. The architecture is presented in the figure 3.10.

3.2.3 Encoder $q(z|x)$

The encoder network is an RNN as described in the section 3.1.2. For each timestep is fed the corresponding element of the input and the last hidden state is fed to the stochastic layer. This input fed to the stochastic layer has to be representative of the whole input sequence.

Indeed, the information representing the whole sentence needs to be transmitted to the stochastic layer because the model is required to reconstruct the complete sentence which is impossible to achieve without transmitting a complete representation of the sentence through the model.

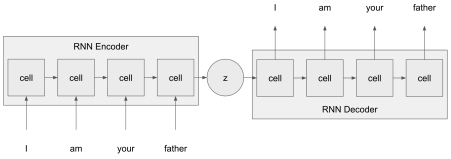


Figure 3.10: The VRAE is similar to the VAE model except that the encoder and decoder networks are implemented using Recurrent Neural Networks

The summarization of the input sequence is operated in a step-wise fashion. At each time step, the model is fed with both the previous hidden state and the corresponding element from the input. Thus, the current output is a representation of the information contained in the previous hidden state and the current input. Following this line of thought recursively, we build this overall representation step by step.

3.2.4 Decoder $p(x|z)$

The decoder network is an RNN as described in the section 3.1.3. The decoder receives the samples from the stochastic layer as input and produces an element of the sequence at each step.

3.2.5 The objective Function

The objective function is similar to the Variational Lower Bound introduced in the section 2.2.2 in the context of the VAE framework: we optimize the lower bound of the expected value of the reconstruction of x. As a reminder, this term is decomposed into two terms: the regularization term which shapes the latent space to a uniform Gaussian and a reconstruction loss which evaluates the quality of the reconstruction. As we are working with sequential data, the reconstruction loss may be a reduction of more basic measures applied to each sequence element.

3.2.6 Training the model

Similarly to the VAE framework, the model is trained using Stochastic Gradient Descent (SGD). Gradients are computed automatically by Tensorflow. Thanks to the re-parameterization trick and because unfolded RNNs behave like a feed-forward neural network, the gradients can be propagated at every point of the graph.

Given the complexity of the model, we use a GPU acceleration to speed up the process. First experiments have shown that the use of the GPU seepds up the training process from 10 to 20 times.

3.3 Additional Improvements

Training the Variational Auto-Encoder was more difficult than training a deterministic auto-encoder because the stochastic nature of the representation and the constraints on the latent representations step up the complexity of the problem. Furthermore, training very deep neural networks is known to be a difficult task which is evenly true for recurrent neural networks which can be seen as deep architectures in their unfolded form. This is why it is not surprising to see the vanilla Variational Recurrent Auto-Encoder fail on the task of modeling complex data structures such as time series: we are training a remarkably complex structure into an especially demanding framework.

In this section, we will introduce some additional improvements to the architecture previously presented in this chapter. Thus, this section is a listing of different features which can be added independently to the actual framework. it is important to notice that most of these features are commonly used in the machine learning communities. However, these features are not explicitly stated in the research papers which explain that state-of-the-art results are often difficult to obtain.

Last, because some features are considered as elementary good practices in the deep learning communities. Some features have been expected to have a positive effect on the overall performances and have not been evaluated in the context of this project.

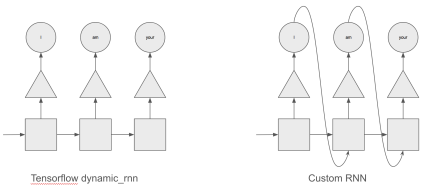


Figure 3.11: The dynamic_rnn class from Tensorflow implements the vanilla RNN (left): cells are simply fed with the previous hidden states. By contrast, our custom implementation of the RNN (right) couples the RNN cell with a projection layer which produces an approximation of the true output. this predicted output is used as input for the next RNN cell which can use this knowledge to make better predictions. The squares represent the RNN cell, the triangles represent the projection layer and the circles the outputs

3.3.1 Decoder: Feeding back the predicted Values

In the previous sections, we stated that the decoder model was implemented as a recurrent neural network. However, the implementation of the dynamic RNN available in Tensorflow is designed such as the RNN cells transmit only the hidden state between time steps.

However, this model can be improved because, in the context of the vanilla RNN, predictions are made using a representation of all the previous steps. Nonetheless, most recent time steps are often more helpful to predict the next value than the older ones. For instance, when generating text, knowing that the last character is a 'v' is enough to guess that the next character is a vowel. Thus, fostering the transmission of information related to the last step could greatly reduce the complexity of the decoding task.

Projection Layer In the context of a decoding model, each RNN cell predicts one element of the output sequence. Depending on the structure of the data, we are predicting either a value in a range (regression problem) or particular class (classification problem). Thus, the raw output of the RNN cell must be converted to fit the output's requirements. In response to this problem, a projection layer is used. the projection layer acts at two levels. First, it adapts the dimension of cell's output to the dimension of the true output (i.e. the dimensionality of a data point within the sequence or the number of classes). Second, this output needs to be normalized because, in the case of a classification problem, the output over each dimension corresponds to a likelihood for the element to belong to each class. In the case of a regression problem, we are most likely to wish our output to lie in $[0, 1]$. Therefore, in the case of classification task such as predicting a specific character, the projection layer will be defined as follows:

$$y_t = Softmax(W_{proj} y_{raw_t} + b_{proj})$$

In the case of a regression problem, we can simply use a linear layer and additionally normalize the output using an activation function.

Last, this is important to notice that this projection operation is the same for every time step. Thus, only one set of parameters must be defined.

Feeding back the Predicted Values We have seen that a decoder can be improved by using an approximation of the preceding data point. In practice, this is done by coupling the projection layer with the RNN cell and by adding a connection from the output of projection layer $t-1$ to the input of the RNN cell t. In this way, the RNN cell can directly use an approximation of the previous element. This wiring scheme is shown in the figure 3.11 and shows the difference with the vanilla decoder.

Implementation To take profit from the dynamic capability and the gain of performances provided by the use of Tensorflow's API, we are required to use this API. Implementing the vanilla RNN with Tensorflow is straightforward using the *dynamic_rnn* class. Unfortunately, this class does not allow us to feed back the predicted values.

Fortunately, Tensorflow's API is quite flexible and features a class *raw_rnn* that allows to implement custom operations and a custom wiring scheme between the cells using a customizable callable $loop\_fn$. The implementation follows this line:

Listing 3.3: loop_fn

```
1
def loop_fn(time, cell_output, cell_state, loop_state):
    emit_output = cell_output  # == None for time == 0
    prev_out = cell_output
    elements_finished = (time >= num_steps)
    finished = tf.reduce_all(elements_finished)
    if cell_output is None:  # time == 0
        next_cell_state = cell.zero_state(batch_size, tf.float32)
        next_input_value = tf.concat([inputs_ta.read(time),tf.zeros([batch_size,data_dim],dtype=tf.
            ↪ float32)], 1)
    else:
        next_cell_state = cell_state
        next_input_value = tf.cond(
            finished,
            lambda:tf.concat([ tf.zeros([batch_size,state_size], dtype=tf.float32), tf.add(tf.matmul(
                ↪ prev_out, W_proj), b_proj)], 1) ,
            lambda:tf.concat([inputs_ta.read(time), tf.add(tf.matmul(prev_out, W_proj), b_proj) ], 1) )

    next_input = tf.cond(
        finished,
        lambda: tf.zeros([batch_size, data_dim + state_size], dtype=tf.float32),
        lambda: next_input_value )
    next_loop_state = None
    return (elements_finished, next_input, next_cell_state,
                    emit_output, next_loop_state)
```

Evaluation We observed that feeding back the previous projected outputs significantly helps the model. In practice, it has been observed that this technique makes the model more robust and accelerates the training. The figure 3.12 shows the evolution of the objective function over time for two problems distinguished by the use and non-use of this technique only.

3.3.2 Decoder: Teacher Forcing

Using the output at $t-1$ to facilitate the prediction at the step t improves the decoder model. However, during the early stage of the training, the predicted output is very likely to be false. Therefore the model is most likely to be trained using wrong predicted previous steps and will tend to learn incorrect relationships since the value of the predicted outputs will change with time. As a consequence, the

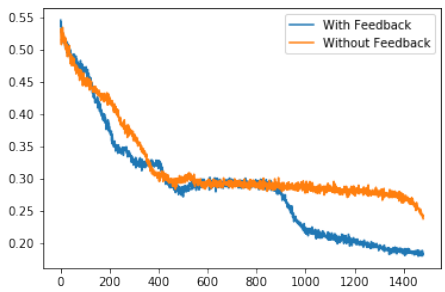


Figure 3.12: Feeding the projected previous output (the expected values) makes the model more robust and makes it converge faster

model will be required to learn new relationships while the predicted output is corrected over time and this will most likely slow down the process.

[WZ89] introduces a technique to speed up the training process. Instead of feeding the previously predicted output, the cell is fed with the true previous output during training. In this way, the model can learn directly to predict elements using the true information while the model is not yet robust enough to produce it by itself.

To measure the effectiveness of this technique, we trained similar models on a single language modeling task (character level) with sentences from 8 to 30 characters. We observe that teacher forcing accelerate the training process (figure 3.13). Moreover, the quality of the reconstruction obtained with teacher forcing seems conductive to a better quality. Few examples of reconstructions are shown table 3.1. We can observe that the model trained with teacher forcing is better at reconstructing existing words while the other model tends to produce meaningless chains of characters.

This technique, however, presents some drawbacks. Indeed, [Chi+17] points out the trade-off between short term and long term accuracy. In this paper, the goal is to build a model able to play video games by taking only the matrix of pixels as input. In order to take a decision, such model must be able to simulate the next frames. This paper shows that using teacher forcing increases short-term accuracy but affects negatively the ability to model long-term dependencies.

Regarding the language modeling task, we will choose to use teacher forcing because our experiment will be limited to short sentences. Moreover, regarding the language modeling task, the first quality assessment judges the ability of the model to form real words instead of forming meaningless chains of characters. In this context, teacher forcing is a good choice since it fosters short term accuracy (ie. word reconstruction).

ground true	teacher forcing	no teacher forcing
what does neil simon do?	what does he lane teer?	what moee coue ee do?
minor spoilers ahead.	monor spoilers ahead!	lonee peeele teat.
again, i've done research.	avoid it like per eraters.	aooon, lave ae crencs
it really was that bad.	it really was that bad.	it really was that bad.

Table 3.1: Reconstruction of a few sentences after 10000 training steps. the model using teacher forcing tend to perform slightly better. For instance, the model with teacher forcing forms some meaningful chain of characters while the other model totally fails (line 3). Similar results are observed on randomly picked sentences

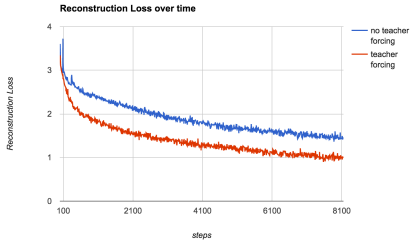


Figure 3.13: Using teacher forcing helps the model to reach higher accuracies

3.3.3 A Solution to the Vanishing gradients problem: LSTM

3.3.3.1 Problem

Recurrent Neural Networks, introduced in section 3.1, are a great architecture to model sequential data since at each step the information of the previous elements can be used to infer the current element. As a consequence, recurrent neural networks are theoretically able to propagate information from early elements to the last.

A basic RNN uses simple a non-linear layer as a cell. A common choice is the sigmoid function or hyperbolic tangent. However, these functions map values from the range of the real numbers to the small range [0,1]. This non-linear layer is fundamental for the cell to learn non-linear dependencies. However, this mapping, by transforming a large input range into a very small range, reduces the ability of the network to preserve information while transmitting it from the first layers to the last. Gradients will be recursively mapped to a smaller range, which makes them progressively vanish.

Indeed, for example, in the case of a pure decoder, we can easily find a sufficient condition on the weights of the layer for the Vanishing Gradients problem. Given s_t the state size, x_t the input, and z_t a variable fed to a basic decoder and ϕ a non-linear function such as the sigmoid function, we have:

$$s_{t+1} = \phi(z_t) \text{ where } z_t = W s_t + b$$

$$\text{Then } |s_{t+1} - s_t| = |\phi(z_t) - \phi(z_{t-1}|$$

$$\text{However}, \forall a, b \in R^2, a \neq b, \exists c \ [a, b] \mid \phi'(c) = \frac{\phi(b) - \phi(a)}{b - a}$$

$$\text{because } \phi \in C^\infty \text{(mean value theorem)}$$

$$\text{Let set } c \in [z_t, z_{t-1}] | \phi'(c) = \frac{\phi(z_t) - \phi(z_{t-1})}{z_t - z_{t-1}}$$

$$\text{Then we have } |s_{t+1} - s_t| = |\phi'(c)||z_t - z_{t-1}| = |\phi'(c)| \, \|W\| \, |s_t - s_{t-1}|$$

$$\text{Finally, } s_{t+1} = |\phi'(c)| \, \|W\| \, \Delta s_t$$

$$\text{As for sigmoid or tanh } \forall c \in R, \phi'(c) \leq 1$$

$$\Delta s_{t+1} \leq \|W\| \, \Delta s_t$$

$$\text{By recurrence } \forall k, \Delta_{t+k} \leq \|W\|^k \, \Delta_t$$

$$\text{Therefore if } \|W\| < 1$$

$$\text{Then } \lim_{k \to \infty} \Delta s_{t+k} = 0$$

This is why basic RNN cells are incompatible with long-term dependencies: when sequences become longer, information vanishes progressively.

This problem of long terms dependencies prevents us from building more advanced models. For example, in the sentence "The trees are green", it is pretty obvious that the last word to be predicted is "green" because know that trees are always green, therefore we don't need other contextual information to make this prediction. The word "tree" was enough to predict "green" and the dependencies are really

short. However, if we try to predict the last word in the sentence "I used to study music when I was younger, that's why I can play music.", then we need to access information about the context contained in the words "studied music" which are placed far away from the last word. Therefore the task becomes much more difficult for our model because it has to use the information related to "study music" which is diluted with other information representing the beginning of the sentence: "I used to study music when I was younger that's why I can".

3.3.3.2 Concept

A solution to this problem is the Long Short-Term Memory (LSTM) cell introduced by [HS97]. The LSTM is a type of RNN cell which is able to control what it reads, write and delete from the hidden state. The cell doesn't morph this information by shrinking it into a small space but adopts an additive process with which information is changed incrementally.

Indeed, LSTM cells have the ability to learn how to explicitly select what to write and what to forget. Using this technique, the network doesn't learn to transform a hidden state with a non-linear transformation. Instead, the network learns to take binary decisions to write or delete information. In the previous section, we have observed that the sequential application of the cell transformation was causing the Vanishing Gradient Problem. Thus, the binary reading and writing mechanism keeps the information flow from this decaying process observed with vanilla RNN.

The LSTM cell has to be compatible with gradient descent and back-propagation. Thus, this mechanism uses a system of gates which are themselves controlled by differentiable functions. A gate can simply be represented as the multiplication by a scalar from 0 to 1. If the gate is set to 0, no information is passed. If the gate is set to 1, the information can go through. This can be simply implemented as a non-linear layer which is compatible with usual optimization algorithms.

3.3.3.3 Implementation

Using standard non linear layers as cell for a recurrent neural network, we define the hidden state at each timestep t such as:

$$s_t = h(Ws_{t-1} + Ux_t + b)$$

where s_t is the current state, s_{t-1} the previous state, x_t is the current input, W and U are weights, b is the bias and h is the activation function which is commonly a non-linear function such as tanh or $ReLU$. For this kind of RNN cell, we output directly the hidden state.

By contrast, the LSTM cell adopts a different mechanism in order to choose what information to transmit or not. This transmission mechanism is controlled by 3 different mechanisms:

- write: write down the information to the memory (i.e.: transmit the information to the next cell)

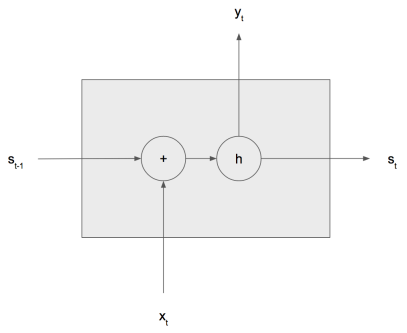


Figure 3.14: A basic RNN cell: a simple non linear layer

- read: read the information from memory (i.e.: read the information from the previous cells)
- forget: delete a part of the information (i.e.: do not transmit an information received from the previous steps)

In order to control these mechanisms, the LSTM cell uses a system of "gates". The gates let the information flow or not depending on their own values. Mathematically, we express the gates as following with independent weights and biases:

- the input gate $i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$ for writing
- the output gate $o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$ for reading
- the forget gate $f_t = \sigma(W_f s_{t-1} + U_f x_t + b_f)$ to forget information

Where W and b are weights and biases, $\sigma is the activation function (sigmoid)$.

Besides these gates, the LSTM cell uses not only one output: the hidden state h_t but also a cell state C_t which acts as a writable memory.

At each timestep, a memory candidate $C'_t = \tanh(h_{t-1} + x_t)$ is created inside the cell and will be added to the cell state C_t depending on the value of the input gate i_t. If i_t is 1, then the candidate C'_t is fully written to the memory (the previous cell state) C_t, itself controlled by the forget gate:

$$C_t = f_t * C_{t-1} + i_t * C'_t$$

On the other hand, the output of the cell is controlled by the output gate. Indeed, the output of a current cell is a transformation of the current state of the cell (memory). Following this line of thoughts, we define:

$$h_t = o_t * \tanh(C_t)$$

The full cell is shown in the figure 3.15. The LSTM cell is trainable with SGD and is fully compatible with usual deep learning architectures.

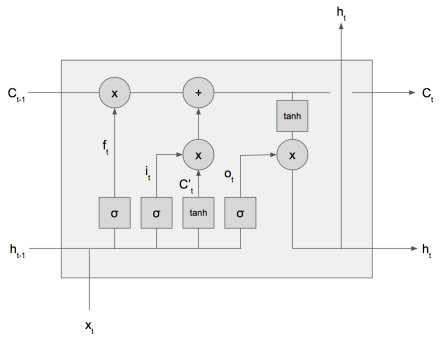


Figure 3.15: Inside an LSTM cell. An LSTM cell is able to learn what to read, write or delete in the memory C_t. The values i_t, o_t and f_t control the input, output and forget gates which control this mechanism. At each timestep, a memory candidate C'_t is added or not to the previous memory C_t depending on the output gate. Thanks to this architecture, the LSTM cell allows to construct RNNs with longer dependencies

3.3.4 Deterministic Warm-up and Weak Regularization

Training VRAE is a difficult task because of the complexity of the model and the presence of two terms in the loss function which can be competing against each other during the training phase. Indeed, one of the terms fosters the accuracy while the other regularize the model. It is not obvious that improving the accuracy improve the regularization and the other way around.

The first trials with the VRAE model used on sequential data (sinusoids) failed because the model was unable to converge into a state with which a good reconstruction performances were obtained without damaging the regularization quality. During training, we have seen the regularization loss converging fast while the reconstruction loss remained stagnant: minimizing the regularization loss provided more outcomes than minimizing the reconstruction loss and both processes seem to be antagonist. As a result, the model had good regularization qualities but was only able to generate zero filled vectors.

In order to explain why both terms are adversarial, let's focus on the objective function. As a reminder, we want to maximize the Variational Lower Bound:

$$L \geq L_v = -D_{KL}(q(z|x^{(i)})||p(z)) + E_{q(z|x^{(i)})}(\log p(x^{(i)}|z)$$

As a reminder, this lower bound can be expressed as two independents terms : latent loss L_l and the reconstruction loss L_r with $L_v = L_l + L_r$:

$$L_l = -D_{KL}(q(z|x^{(i)})||p(z))$$
$$L_r = E_{q(z|x^{(i)})}(\log p(x^{(i)}|z)$$

The reconstruction loss L_r evaluates the reconstruction quality of the input. The latent loss L_l, however, acts as a regularization term and ensures that prior distribution is shaped as Gaussian distribution. In practical terms, it forces the model to distribute all the inputs in the latent space such as they form a Gaussian distribution.

On the one hand, the inputs are represented in the latent space not as a single point but as a Gaussian distribution with a non-null width. On the second hand, pushed by the effect of the regularization term, the model tries to occupy the whole space to fit the target distribution $p(z)$. As a result, the model tries to encode inputs x into a *large* Gaussian distribution in order to fill-up the whole space. Consequently, samples z drawn from the posterior distributions $q_\phi(z|x)$ are randomly distributed in an area that grows as the regularization term is minimized and the parameter σ becomes bigger. Regarding the generative model $p_\theta(x|z)$, the decoding task becomes more and more difficult because samples z becomes noisier and noisier. The difficulty of the decoding tasks increases as the regularization term is fostered and the model indefinitely fail to favor the reconstruction term. As a result, the model fails to find a reasonable minimum. Two solutions help us to overcome this problem:

1. Weakening the Latent loss by simply adding a weight to this loss. This weight will be called α_l and noted *latent_loss_weight* in the code. In this way, the model is pushed to prioritized the reconstruction cost because it returns a bigger outcome[3].

2. Deterministic warm-up, which has been proposed by [Kaa+16], helps the model to focus on reconstruction during the early steps of training by starting with a null latent loss and adding progressively this term over time. The weight controlling this deterministic warm-up will be called β and noted B in the code.

To sum up, both techniques taken into account, we optimize:

$$L'_v = \beta\alpha_l L_l + L_r, \beta \leq 1, \alpha_l \leq 1$$

Using this technique, we are now able to train the model and obtain a reasonable reconstruction accuracy.

[3] this technique was implemented in the original paper [Fv14]

3.3.5 Dynamic RNN

Tensorflow is originally made to create a static graph before processing. For Recurrent Neural Networks, the network is unfolded during the construction of the graph with a fixed number of steps. For example, 20 units will be unfolded to process sequences of 20 elements each. However, while it seems reasonable to use such approach for time series since we set a fixed number of samples to use, it is less reasonable to do so for natural language. Indeed, sentences may be of an average length of 20 words in our dataset. However, we will also find much longer sentences up 100 words. In this case, if we wish to process the whole dataset, we will need to unroll a static RNN composed of 100 steps which is an important waste of memory and computing power.

A first solution, as used in the earliest version of the sequence2sequence model, was to split the data into buckets of same sentence size. For example, sentences from 5 to 20 words were put together in a bucket and sentences from 20 to 40 words were put together in another bucket. Then, different RNN were constructed to process the different bucket.

In the last releases, Tensorflow has been updated with a dynamic implementation of the RNN: the graph is now unfolded on run time which ensures to unfold the minimum number of units and save computational resources. However, this forces us to use the Tensorflow API which makes the implementation slightly more difficult, especially to implement some techniques such as the projection feedback as described in the section 3.3.1 or teacher forcing.

3.3.6 Learning Rate Decay

Setting the learning rate value is a trade-off between robustness and convergence speed. We would like this value to be high to save time: the model will converge faster. However, a model will fail to converge if the learning rate is too high because the parameters will change abruptly and the model will be likely to miss a local minimum. This effect is represented in the figure 3.16.

Scheduling the learning rate is a good solution speed up the training in the early stage while keeping a slow approach when fine tuning is required. The idea is simple: we use a high learning rate in the early steps and decrease this value progressively. We choose to decrease the learning rate if the loss stays constant after a given number of steps.

3.3.7 Reversing inputs

According to [Cho+14], reversing inputs can lead to a better utilization of the LSTM memory and lead to better results. Indeed, for sequential data such as the language, the information contained in the first elements of the sequence is often more valuable to predict the whole sentence than the last element.

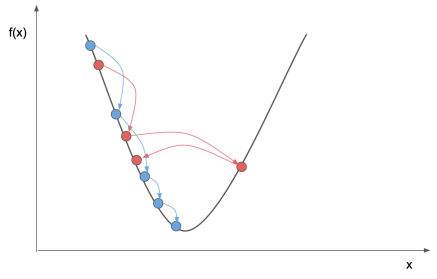


Figure 3.16: The optimization algorithm fails to find a minimum if the learning rate is too high. For this example, we try to find the minimum of the function f (gray curve). In red are shown the steps of the optimization algorithm with a high learning rate. In blue is shown the steps with a smaller learning rate

For example, given the sentence ”roses are red”, it seems easier to predict the last word ”red” given the first word ”rose” than it is to predict the first word ”roses” given the last word ”red”.

However, we have seen previously that our model uses Recurrent Neural Networks to encode sentences. With RNNs, the information contained in the first element of the input has to go through the whole unfolded network to reach the last input which is connected to the stochastic layer. However, even if the LSTM cells presented in the section 3.3.3 increase the ability of the network to transmit information, this information is still likely to be lost.

Reversing the input is a simple yet effective solution to this problem. Indeed, first elements of the sequence are now placed at the end of the RNNs. In this way, the information representing the first elements of the sentence can be transmitted more easily to the stochastic layer which means a better transmission to the decoder. Since the decoder has a better access to the first elements of the sequence, those can be decoded more easily.

3.3.8 Bidirectional RNN

In the previous section, we introduced a simple trick consisting in reversing the input sequence. This was motivated by the fact that the last elements of a sequence are encoded more easily since they are less likely to be transformed by other RNN layers. While the first elements of a sequences are more useful to the decoder, other elements are still valuable. Therefore an easy way to encode more information would be to use two RNNs side by side: one would encode the sentence in one direction and the other would encode the sentence in the opposite direction. Using two encoders, we benefit from both encoding directions.

This is the motivation for the bidirectional RNN introduced by [SPG97] and represented in the figure 3.17. The bidirectional RNN is an RNN with forward and backward connections. The network uses two distinct cells which process information respectively in positive and negative direction. This is particularly useful to encode long sequences since the difficulty to encode first elements increase as the length of the sequences increases. In practice, it has proven to be more efficient than the unidirectional RNN as it is supported by the experiment presented in the figure 3.18.

3.3.9 Dropout

Large neural networks are prone to overwriting: instead of learning general rules about the structure of the observed data, the model learns specific examples. in the field of machine learning, a common approach to avoid overfitting is to use Bagging (Bootstrap Aggregation) which improves stability, accuracy and reduce variance and overfitting. In bagging, multiple models are trained on randomly chosen subsets. The final model is obtained by averaging the sub-models.

This technique is doable with traditional machine learning models since their complexity is relatively low in comparison with nowadays hardware. However, the application of bagging to deep learning

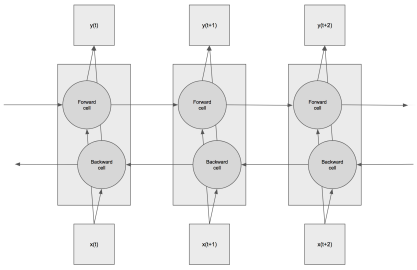


Figure 3.17: The bidirectional RNN process information in both negative and positive directions. This allows us to predict elements using past and future information. When used as an encoder, the bidirectional RNN is useful to keep a good encoding of both the begining and the end of a sequence

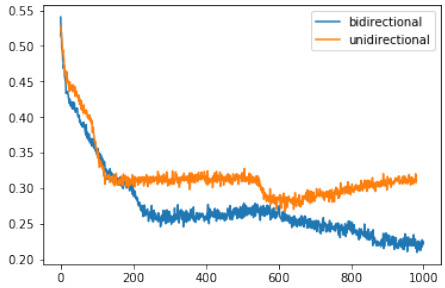


Figure 3.18: Using a bidirectional rnn as encoder, the model conerges faster than using a simple rnn as encoder. The y axis corresponds to the reconstruction loss. In this experiment, the RMSE is used

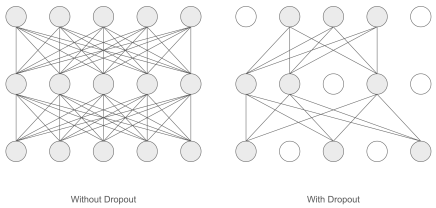


Figure 3.19: Dropout acts as a regularization technique for the neural networks by imitating the Bagging technique. At each training step, a randomly chosen set of units is muted. Thus, different versions of the same model are trained on virtually different training sets. At test time, however, all units are kept a produce an average result of the sub-models used during training

models seems unrealistic because of three reasons. First, each model would need to be fined tuned and hyper-parameters tuning is an exhausting task. Second, deep learning models are computationally expensive thus the number of sub-models is bound to the hardware availability. Third, deep learning models require to be trained on very large datasets and available datasets are often limited by their size.

[Sri+14] proposes a simple yet effective alternative solution to adapt such technique to the neural networks. This technique doesn't involve the use of independent models, hence it solves the problem of hardware requirements and independent hyper-parameters tuning. The dropout technique virtually combines a high number of different architecture by randomly muting part of the neurons during training. At each training step, a random set of units is "dropped out" which provides us with a different architecture at each time step. A simple way to drop units is to set their connections to zero. Usually, the probability of keeping a unit is chosen to 0.5 for output units while it is chosen close to 1 for input units. At test time, however, all units are kept. Because of the linear combinations between units, we obtain an averaging of the sub-models models used during training.

Regarding the Variational Recurrent Auto-Encoder, we wish to apply dropout to the cells within the recurrent neural networks with the aim to regularize the model and improve generalization. A better generalization would mean a better ability to model higher level language features. The implementation of the dropout with Tensorflow is straightforward using the module DropoutWrapper.

3.4 Experiment: Modeling Timeseries

3.4.1 Motivations

The aim of this project is to build a generative model of the language using the Variational Autoencoder framework. However, working with natural language is a difficult task by itself because of the complexity

of the structure of the data. Moreover, it requires important computational resources, therefore, it would slow down the development because each validation experiment would require hours to run. On the other hand developing the model itself is a difficult task because it requires dealing with advanced deep learning techniques.

This motivates the choice to work with a toy experiment at first. Time series are a good candidate because results can be assessed visually and because we can choose a limited set of functions which are easy to model.

3.4.2 Data

Time series are sequences of values indexed on an ordered time range. The recorded value can be a specific exchange rate, the gas price or the height of ocean tides over time. Time series are interesting because they often present a recurrent behavior. For example, the height of the ocean tides have a cyclic behavior related to the moon phases and general variation patterns can be observed with exchange rates because the markets tend to adopt similar behaviors in reaction to specific events.

Regarding this experiment, we simulate our own set of time series using sinusoids. Sinusoids are well fitted to evaluate the usability of Recurrent Neural Networks within the VAE framework because it is a simple function to model with RNNs. Nonetheless, because we are dealing with a generative model, we wish not only to model a single sinusoid but a family $\mathscr{F}$ of sinusoids defined by different phases and frequencies. In order to do so, the functions $f \in \mathscr{F}$ depend on two continuous variables x and y:

$$f : t \in R \rightarrow sin(\pi x t + \pi y))$$

Finally we can define the ensemble of functions F:

$$\mathscr{F} = \{t \rightarrow sin(\pi x t + \pi y), (x, y) \in [-1, 1]^2\}$$

The next step is to generate timeseries from the family of functions $\mathscr{F}$. For the sake of simplicity, we use a fixed time vector named t. Thus, we finally have a family of timeseries $\mathscr{S}_t$ defined as:

$$\mathscr{S}_t = \{f(t), f \in \mathscr{F}\}$$

Finally, the ensemble of observations in $\mathscr{S}_t$ is generated using two uniform variables x and y. A visualization of the dataset is presented in the figure 3.20.

3.4.3 Implementation

The model described in section 3.2 and the improvements proposed in section 3.3 have been implemented using Tensorflow. The whole code is presented in a Jupyter Notebook available the appendix B. The loss function used for the reconstruction term introduced in the section 3.2.5 is the RMSE (root-mean-square error) which measures the average point-wise distance between the true output and the predicted output:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{n} (\hat{y}_t - y_t)^2}{n}}.$$

The mode has been implemented using Tensorflow and its API which features high-level functions such as an implementation of a dynamic recurrent network and LSTM cells. However, other parts such as the decoder have been implemented manually using lower level modules such as the module *raw_rnn*.

The time vector t was fixed to a length of 40 elements, the model was configured with the following parameters:

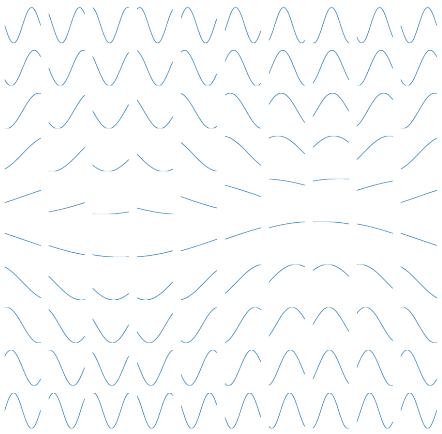


Figure 3.20: The time series are generated from a family of functions $\mathscr{F}$ and fixed time vector t. The family of functions $\mathscr{F}$ is generated from two continuous variables x and y. On this graph are represented a few samples of these time series. Each sample corresponds to a different configuration of the variable (x,y): the Y axis controls the frequency and the X axis controls the phase. The goal of this experiment is to model this ensemble of sequences using the VRAE

```
   1
1  num_steps = 40               # length of each record
2  batch_size = 256
3  state_size = 60              # RNN Cells state size
4  learning_rate = 0.0005
5  epoches = 5000
6  beta_decay_offset = 300      # Deterministic Warm up
7  beta_decay_period = 1000     # Deterministic Warm up
8  latent_loss_weight = 0.03    # latent loss weight
9  data_dim = 1                 # dimension of the data
10 latent_dim = 2               # latent space dimension
```

3.4.4 Results

Given a fixed time vector t, the model has successfully learn the space of observations $S_t = \{sin(\pi xt + \pi y), (x, y) \in [-1, 1]^2\}$ with an acceptable reconstruction error. Furthermore, the qaulitative analysis of the results indicates good performances as well. A few samples are presented in the figure 3.21. However, the most remarkable result of this experiment is the ability of the model to find a latent representation of this set of observations. While the model was generated by controlling the phase x and the frequency y, the two dimensions of z seem to control different features. This can be explained by the fact that the model doesn't learn the whole sinus function but a limited set of observations restricted by a fixed time range. Thus, the concept of frequency and phase is quite irrelevant.

the reconstruction of sequences is not the most interesting result because a deterministic model such as the encoder-decoder framework presented in the section 3.1 would have been perfectly fitted to this reproduction task. The most interesting point is the generative nature of the model: for any z from the latent space we can construct a time series $\hat{x}$ with similar properties to the initial specimen. Most interestingly, we observe that the model has built a smooth representation of the data set: similar items are represented with similar latent codes.

3.5 Conclusion

Sequential data are often complex to model but their structure can be used to lower the complexity of the task. Indeed, with sequential data, previous elements can help the model to predict a current

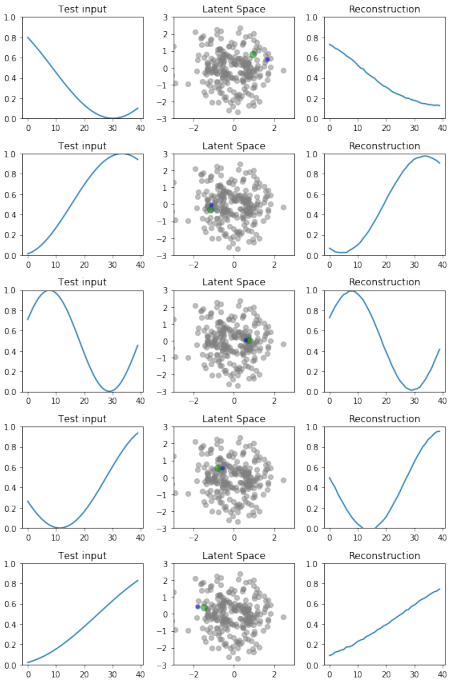


Figure 3.21: The VRAE model learns by reconstructing observations. The left column shows the inputs fed to the model, the second column shows the position of the inputs in the latent space and the third column shows the reconstruction. We can observe that the model performs a good regularization (the latent space is Gaussian-shaped) and provide good reconstructions

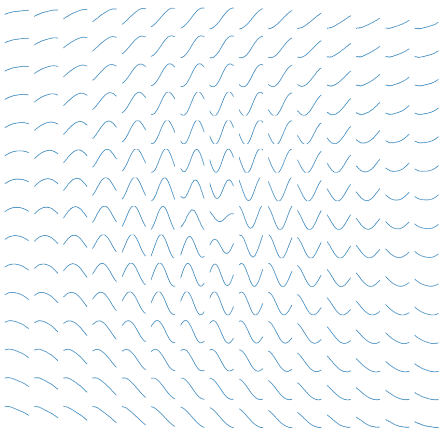


Figure 3.22: The following plot represents the latent representation of the set of observation by the model. We sample $(z_1, z_2) \in [1.5, 1.5]$ from the latent space and draw their reconstruction. Each point (x, y) in the grid corresponds to a different combination (z_1, z_2). The model has built a smooth latent representation of the ensemble of observations

element. Therefore instead of predicting a full sequence, we can predict each element one after another using the past information. Recurrent Neural Network is a special case of a neural network which is based on this property. A unique cell is used at each time step to predict the current input based on the previous inputs.

In this chapter, after having introduced recurrent neural networks, we have introduced the Variational Recurrent Auto-Encoder framework: a deep generative model built on the top of the Variational Auto-Encoder framework. This is basically a VAE model which uses recurrent neural networks as encoder and decoder to tackle problems involving sequential data.

With additional improvements (LSTM, dropout, etc..), the VRAE model performed well on the task of modeling an ensemble of time series. The model was not only able to reconstruct each of the input with an acceptable error but it was also able to find a structured latent representation of the ensemble of observations. The model is now ready to face the natural language modeling task.

CHAPTER 4

Generating Sentences from a Continuous Space

In the previous chapter, we introduced the Variational Recurrent Auto-Encoder. A generative model fitted for the modeling of sequential data. The aim of this chapter is to apply this model to the natural language modeling task, a highly challenging task on three levels:

First, it is required to model a very large vocabulary. As a reference, the *English Oxford Dictionary* contains 171,476 words. A large number which can be increased by the existence of proper nouns, slang and words that exist in different forms such as the verb *to be* (I *am*, you *are*, he *is*). Second modeling language requires mastering its grammar: a complex set of rules which is often difficult to master even for a human being. Third, natural language is also based on a set of informal rules which depend on the context and require to use the right linguistic style.

In this chapter, we will investigate the use of the Variation Recurrent Auto-Encoder with regard to this task. First, we will present how to encode language, then we will motivate the choice of a limited dataset and we will last analyze the performances of the model with regard to the natural language modeling task.

4.1 Character-Level Encoding

We chose to use a character-level encoding to build the language model. The next section will present the underlying motivations behind this choice.

4.1.1 Characters versus Word Level Encoding

Natural Language Models are based on Recurrent Neural Networks which require being fed with units of language iteratively. The unit of text can be chosen as the word as used in [Cho+14], the character as in [Kim+15] or even sub-words units as in [Wu+16].

For each approach, a vocabulary needs to be defined. For example, in the case of the character-level models, the vocabulary can be simply the alphabet. However, in regard to word-level models, the vocabulary has to be limited and rules to limit this vocabulary have to be defined.

As a result, character-level models are attractive for their generalization capacity because they are independent of a fixed vocabulary. On the contrary, the need to use a word-based vocabulary is the key weakness point of the word-level models.

First, the use of a static vocabulary necessarily exclude some words. Therefore we need to define rules to handle unknown words and when missing words are encountered. furthermore, we can't guarantee to obtain good results because the network will make an inference based on the unknown when unknown words are encountered. Moreover, a word-based model is prone to errors when misspellings are encountered because misspelled words are considered as unknown words. By contrast, a character-level model is more likely to handle unknown words and misspellings because the structure of the words is likely to be similar to known words at the character level.

Second, using a vocabulary require applying the softmax operation on a vector of the dimension of the vocabulary which is extremely costly. Thus, using a character-based vocabulary tends to be a good solution for this project because of the limited hardware availability.

Third, [RJS17] presents state-of-the-art results in sentiment modeling using a character model. Thus, it seems that this model is well adapted to the purpose of this project.

Nevertheless, using character-based encoding increases drastically the sizes of the encodings of the sequences. For instance, the sentence *"I hate this movie."* can be encoded into 4 words but is encoded into 18 characters. In the section 3.3.3, we introduced the vanishing gradients problem. Even if the LSTM cells are a partial solution to this problem, encoding long sentences remains an important challenge. As a consequence, a project focusing on long textual structures would foster the use of a word-based solution.

4.1.2 Practical Considerations Behind the Choice of a Character-level Model

First, this project has not been done in an industrial context and is done with limited computing resources. Second, this project aims to provide a proof of concept for the transfer of stylistic linguistic features. Thus, the main objective is not to build a model which can handle any sentence including the long ones and focusing on small sentences is a viable choice. Third, the Large Movies Review dataset is made from online reviews written in a non-professional context which explains that misspellings are very likely to occur.

These reasons motivate the choice to build a model at the character level. Furthermore, it is also very interesting to build a model based only on characters because this approach reduces the prior knowledge provided by human hand to the model.

4.1.3 One-Hot Encoding

Sequences are encoded at the character level using a one-hot encoding. Each character is encoded into a vector of the size of the vocabulary (alphanumeric symbols and additional characters). A one-hot encoded vector is a vector of 0s with a 1 corresponding to the encoded character in the vocabulary.

An example is shown in the figure 4.1. The one-hot encoding operation is performed by Tensorflow at runtime.

4.2 Dataset

The aim of this project is to build a generative model able represent the language in two separate components: meaning and linguistic style. Ideally, we would like to have a dataset as large as possible

	a	b	c		b	c
_	0	0	0	1	0	0
a	1	0	0	0	0	0
b	0	1	0	0	1	0
c	0	0	1	0	0	1

Figure 4.1: The sentence "abc bc" is one-hot encoded with the vocabulary " ,a, b, c". Each character is encoded into a vector of the size of the vocabulary. A sentence of 10 characters is encoded into an array of size <sentence size> x <vocabulary size>

to increase the likelihood to cover all the intrinsic rules of the language. A good choice would be the WMT dataset[1], a very large public dataset dedicated to the research in statistical machine translation.

However, working with such large dataset increases de facto the complexity of the task and the hardware requirements. Indeed, this would require implementing the model using multiple GPUs which comes with a higher cost and requires more fluency in the use of TensorFlow. Hence, using a smaller dataset is a necessity for such project.

Furthermore, because we aim to model stylistic features, a dataset oriented towards a specific feature is a good choice. Consequently, a dataset of reviews such as movie reviews or food reviews is a good choice because the language used in the reviews present many differences in terms of negativity and positivity and because the field of language is limited.

This positivity and negativity is often labeled as *sentiment* in the machine learning community and has been widely studied because it is a critical component for customer services nowadays. As a result, many datasets for sentiment analysis are available and tools to measure sentiment are at disposal.

For this project, we will choose to use the Large Movie Review Dataset ([Maa+11]), a dataset designed for sentiment analysis purposes. The dataset contains 50,000 movie reviews with ratings (from 1 to 10). The reviews are split into 25,000 positive reviews and 25,000 negative reviews. Samples are presented in the table 4.1.

4.3 Data Preprocessing

The dataset comes in raw format. Potentially, HTML code is present in the reviews and any character can be present. In order to remove HTML elements, we use the Python package Beautiful Soup [2]. Characters different than alphanumeric characters and (”’.!,?”) are removed using regular expression. Finally, digits are normalized to 0. For example, The string ”1999” will be converted into ”0000”.

Modeling natural language and the difficulty increases with the length of the text elements to model. To reduce the overall length of each data entry in our dataset, we split reviews into sentences and we use sentences as separate items.

Sentences are encoded at the character level as explained in the section 4.1.1. Thus, the model is fed with list of integers and produces list of integers.

Furthermore, deep learning models are computationally expensive to train and this cost increases with the length of the structures we use because each additional character requires the addition of a corresponding RNN cell on run time. This is why we limit the size of the sentences to 45 characters during the experiments. This number was defined as a compromise between the length of sentences, the number of sentences in the dataset and the time required to train the model.

4.4 Implementation

In this section, we will apply the model developed in the previous section to model natural language. The model as presented in the previous section was compatible with this task. Only auxiliary functions for data pre-processing, training management and a specific cost function had to be implemented.

The full project is available on my Github[3].

4.4.1 Practical Matters

For this project, we have chosen to use Google’s open source deep learning framework Tensorflow because of its large and active community. As a result, Tensorflow features a rich API containing high-level

[1] `http://www.statmt.org/`

[2] `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`

[3] `https://github.com/vlievin/VRAE-Tensorflow`

Rating	Review
1	The action in this movie beats Sunny bhai in Gadar. Akshay Kumar possess the superpowers of Leonidus in 300, Neo in Matrix along with Spiderman and Superman. It is hilarious. Except for the typical Akshay Kumar and Anil Kapoor comedy I cannot see anything positive in this film. The story looks like the writer told his 10yr old son to write. The movie is so unreal that Anil Kapoors long range shooting with a shotgun is the least most mistake by the director. Except for the directors Tashan to make this movie there is no other Tashan. I regret wasting my money on this movie and I would not recommend it to anybody. 1/10 is the least I can give on IMDb or I would give it a zero.
2	Much worse than the original. It was actually *painful* to sit through, and it barely held my six year old's interest.Introduction of some new Pokemon is marginally interesting, but storyline is extra-thin, dialogue is still bad, and music is mediocre. Watch the television show instead - it is much better.
9	Geniuses William Cameron Menzies and Herbert George Wells craft this extraordinary anticipation film, with ambition and scope hard to find today. They predict World War II and the way Great Britain was attacked, and also the fact that the war would be followed by a space race. They change the timing; in the film the war and the space exploration are much longer, but there are so many qualitatively correct things that it is amazing. We even see an helicopter (the film is older than them).Unforgettable giant planes and a futuristic meritocracy of scientists that seem Romans with bubble-helmets: if you can see through those funny costumes you may appreciate the state of the art architecture by masters from the 30s, Well's vision of a rationalistic society, interesting reflections on the nature of power, and John Cabal as archetype of the adventurous and inventive human being, the one that chooses to shape reality and not to be shaped by it.9 1/2 out of 10. Inspiring like that final monologue by John Cabal.
7	What could be more schlocky than the idea of private detectives getting involved with the women they're supposed to be spying on? And most of the dialogue as written is perfectly banal.But the actors turn the dialog into something that makes sense. You can see real people behind the unreal lines. And the directing is wonderful. Each scene does just what it has to and ends without dragging on too long.I showed this to several friends in the mid-80s because I was perplexed at how such bad material could be made into such a good movie. The friends enjoyed it too.

Table 4.1: Randomly selected samples from the Large Movie Reviews Dataset

features such as the implementation of the dynamic RNN and the LSTM. Furthermore, Tensorflow is compatible with GPUs, which is a significant asset when considering training a deep and large model.

Regarding the training management, the code presented in the previous section had to be refactorized to be used in a batch fashion: we leave the Jupyter Notebook environment for simple Python scripts because the model will need to be trained for longer periods (from hours to days). Unfortunately, using Jupyter Notebooks requires to be constantly connected to the server, using scripts allows us to run the code in the background. Furthermore, the code has been adapted to be compatible with Tensorboard, a visualization module included in Tensorflow. Tensorboard allows us to visualize the computational graph and different training parameters over time.

4.4.2 Reconstruction Loss

In the section 2.3, we presented the implementation of the Variation Auto-Encoder and stated that the reconstruction loss $\mathcal{L}_r$ is specific to the data structure and the problem we aim to solve. For instance, in the case of the MNIST dataset, we were using the binary cross entropy because we were facing a binary classification problem for each pixel of a given image. In the case of the time series, the binary cross entropy couldn't be used because each data point within a serie had to approximate a scalar within a range [0,1]. Therefore, time series modeling was a regression problem, which justifies the choice of the RMSE as reconstruction term.

In the case of natural language, each sentence is a chain of characters encoded as a chain of integers each of them corresponding to an index of the vocabulary. At runtime, as described in the section 4.1.3, each sentence is one-hot encoded vector.

The Variational Recurrent Auto-Encoder takes each one-hot encoded sentence and generates a reconstruction using the generation model $p_\theta(x|z)$ of the same shape as the the input tensor. The i-th element of the input or of the output of the model can be denoted as follows:

$$x_i = \sum_{j=1}^{M} p_{i,j} e_j$$

where e_j are coordinate vectors and $p_{i,j}$ corresponds to the probability of the i-th character to be the j-th character in the vocabulary. Because we are dealing with probabilities, $p_{i,j}$ must sum up to 1. In the case of the input sequence, the probabilities correspond to the ground true: $p_{i,j}$ is 0 or 1. In the case of the generated sentences, our model doesn't ensure that outputs are summing to 1. Therefore, we will apply the Softmax function to the output [4].

We will name $\hat{x}$ the reconstructed values of x and $\hat{p}_{i,j}$ the normalized predicted probabilities. For each element of the sequence we are facing a classification problem, we use the average cross entropy as loss function:

[4] In this case we use the Softmax function as normalization because characters are mutually exclusive classes

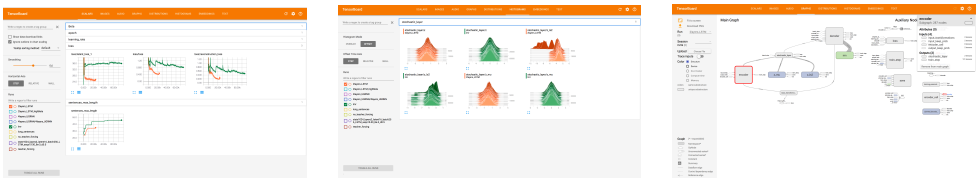


Figure 4.2: Tensorboard is a visualization module included with Tensorflow. It allows us to monitor training parameters and to visualize the computational graph

$$
\begin{aligned}
L_r(x) &= \sum_{i=1}^{N} H(x_i, \hat{x}_i) \\
&= -\sum_{i=1}^{N} \sum_{j=1}^{M} \left(p_{i,j} \log \hat{p}_{i,j} + (1 - p_{i,j}) \log(1 - \hat{p}_{i,j})\right)
\end{aligned}
\tag{4.1}
$$

4.4.3 Hyper-parameters Tuning

Tuning the hyper-parameters of very large models trained with a large dataset is a difficult challenge because training the model takes hours or even days. Generally, 4 different methods can be used to find a good set of hyperparameters:

- *Manual Search*: in this case, we try to use our knowledge and small experiments to guess good parameters
- *Grid Search*: this method consists in searching the optimal parameters using brute-force search. Ranges are defined for each parameter and the performances of the model are evaluated for each combination. This method provides good results but is extremely time-consuming because the model needs to be trained with each configuration.
- *Random Search*: ([BB12]) different configuration of parameters in the same way as the Grid Search method. However, this method differs in the choice of the parameters. Instead of varying parameters linearly in a range, the random search uses different configurations of these parameters within the fixed range. The Random Search method has shown to be more efficient than the Grid Search algorithm for models with many hyperparameters. However, this method is still very time-consuming.
- *Bayesian Optimization*: in this case, the objective function is considered as a noisy function of the hyper-parameters. The main idea consists in using a set of observations (evaluations after training using different parameters) to define a prior of distribution of the objective function conditioned on the hyper-parameters.

In practice, automated hyper-parameters tuning processes are performed using clusters of computers. In this way, experiments can simply run in parallel on different machines. In the context of this project, it seems impracticable to use such methods because of limited hardware availability. For this reasons, we will limit this project to the use of a Manual Search for hyper-parameters and let the implementation of more advanced techniques for further work.

Experimenting with hyper-parameters is a long-winded affair and has required much time and attention during this project. Indeed, multiple parameters can be adjusted which makes the task difficult without any prior knowledge about such model.

A ballpark estimation of the hyper-parameters have been defined by looking into different projects using sequence2sequence models. As a second step, we tried to focus on finding a configuration that works. This task alone took already an astonishingly high number of hours during this project. This is why we will settle for a sub-optimal configuration because the gain of performances seems to decrease exponentially with the invested amount of time. Consequently, it is important to keep in mind that the model presented in this project is likely to be suboptimal.

Finally, the model was trained using the hyper-parameters presented in the table 4.2.

Table 4.2: Hyper parameters chosen for the proposed model

Parameter	Chosen Value	Parameter Description
initial_learning_rate	0.001	initial learning rate
encoder_state_size	1024	State size of the RNN cell of the encoder model
encoder_num_layers	2	Number of layers used for the RNN cell of the encoder model
decoder_state_size	1014	State size of the RNN cell of the decoder model
decoder_num_layers	2	Number of layers used for the RNN cell of the decoder model
latent_dim	16	Number of dimensions used for the latent space
batch_size	256	Number of data points per mini-batch
input_keep_prob	0.9	input keep probability for dropout
output_keep_prob	0.5	output keep probability for dropout
cell	LSTM	type of RNN cell (basic, LSTM, GRU, LNLSTM)
beta_offset	10	deterministic warm-up parameter
beta_period	10	deterministic warm-up parameter
latent_loss_weight	0.001	weight used to weaken the regularization loss

4.4.4 Training

During the early phases of the training, we feed the model with shorter sentences and increase the length progressively. We started with sentences from 8 to 40 characters and increased progressively the upper limit to 40 characters. In total, the model was trained on 34 490 sentences which are considered as a very small dataset for such task. However, training the model using such small dataset was already time-consuming: two days were required to reach a local minimum with an acceptable accuracy. The model was trained using an Amazon ec2 p2 instance with a single GPU during 10 hours.

4.5 Evaluation

4.5.1 Reconstruction

The Variational Auto-Encoder is built on the top of the Sequence2sequence model. Therefore it is first designed to reconstruct the input sentence into the target sentence. Because the model is an auto-encoder, the target sentence is the input sentence itself. As a reminder, the model produces a probability $p_{i,j}$ for each character i to belong to the class j.

$$\hat{x}_i = \sum_{j=1}^{M} \hat{p}_{i,j} e_j$$

In order to produce a human readable output, we need to use this probability to choose a class. We define each predicted character as:

$$\hat{c}_i = \underset{j}{\operatorname{argmax}} \sum_{j=1}^{M} p_{i,j} e_j$$

Cross Entropy First, we evaluate the capacity of the model to perform the reconstruction task on the training set itself and on the testing set to evaluate the ability of the model to generalize. This allows us to have a clear vision on the capacity of the model to learn the structure of a subset of the natural language.

In the context of text generation, the reconstruction loss measures the classification error for each character. The model is trained using the sentence loss as described in the section 4.4.2 which can be considered as the average cross entropy between the true character class and the predicted character

class. After initialization, the model produces random chains of characters and scores an average cross entropy around 3. After training, the model scores an average cross entropy of 0.35 on the training set and 7.41469 on the testing set.

Average Number of Missclassifications Nonetheless, the interpretation of the cross entropy is challenging and gives little hints on how the model performs in practice. Indeed, the average cross entropy has not been chosen for its verbosity but for its mathematical properties (continuity and concavity) which are compatible with SGD. In response to this problem, we introduce a first human readable measure: the misclassification rate:

$$e_{missclassification} = \frac{N_{number\_of\_errors}}{N_{number\_of\_characters}}$$

We report a misclassification rate of 0.03 for the training set and 0.61 for the testing set. Similarly, we come to the same conclusion: the model performs well on known sentences but generalizes very poorly because reconstructions are very dissimilar to the original inputs.

Qualitative Evaluation As a third evaluation, we inspect a few reconstructions. Randomly selected samples are presented in the table 4.3. The model is often unable to reproduce identical sentences from the testing set which is in accordance with the high misclassification rate. Most interestingly, however, in some cases, reconstructions and input sentences are similar with regard to the meaning: instead of producing the same chain of character, the model re-phrase the input sentences while trying to keep the same meaning. For instance, the sentence *"it was a waste of time to watch it."* is interpreted as *"it was a story not worth watching."*: sentences are very different but the meaning remains the same. Other examples of interesting cases of re-phrasing are displayed in the table 4.4. For instance, the model interprets *"It was terrible."* as *"it is worse than the matrix sequels"*: a biased but very meaningful example.

The main objective of this project is to adapt the linguistic style. Adapting the linguistic style requires transforming an input sentence into a version with a different style while keeping the same meaning. In this way, these results are an important step towards this goal because the re-phrasing examples are an adaptation of the linguistic style.

4.5.2 Latent Space

In the context of the Variational Auto-Encoder framework, we aim to build a latent representation of the observed variable x. For simplification purpose, we assumed that $p(z) = \mathcal{N}0, 1)$. The latent term L_l of the loss evaluate the distance from the actual posterior distribution of the model $q_\phi(z|x)$ to the expected distribution $p(z)$:

$$L_l = -D_{KL}(q_\phi(z|x)||p_\theta(z))$$

This term is particularly important in the design of the VAE as a generative model because it ensures that every data point has its latent representation distributed within $\mathcal{N}0, 1)$ and especially that every sample z drawn from $\mathcal{N}(0, 1)$ will be decoded into a correct sentence. Furthermore, the regularization of the latent space is a condition for a continuous representation of the language as described in [Bow+15] because the regularization constraints the model to create a *dense* representation of the language which forces the model to encode similar sentences into a similar z.

Unfortunately, asides from the fact that we obtained a sub-optimal model, it is important to remember that we had weakened the regularization term as explained in the section 3.3.4 . As a result, we can't ensure the latent space to be conform to the ideal Variational auto-Encoder and sentences sampled from some areas in the latent space may be decoded into wrong sentences (random chains of characters, grammar errors, etc..).

Training Set

input	*reconstruction*
it really is that terrible.	it really is that terrible.
do yourself a favor and don't watch.	do yourself a favor and don't watch.
this film is self indulgent rubbish.	this film is self indulgent rubbish.
the writing was trite and shallow.	the writing was trite and shallow.
i know where the maniac lives!	i know where the maniac lives!
bashki is capable of much more.	bashki is capable of much more.

Testing Set

input	*reconstruction*
it was a waste of time to watch it.	it was a story not worth watching.
you wanna cry when the music starts.	you can sue that from the beginning.
overall a disappointment.	overall, a disappointment.
what can i say about ocean's twelve?	what can you say about shomteve soo?
i could rewatch it tomorrow.	i could barely sit through.
the viewer never knows!	the only real oy vey!
the acting was mediocre.	the acting was tery bad.
this film was terrible.	this film was terrible.
next time do ur own thing!	nev see enolt on tve doo?
the vertigo shot!	the best one for me.

Table 4.3: Randomly selected reconstructions from the VRAE model with the original input. The model has learned to reproduce sentences from the training set with a very low error rate. Most interestingly, the model is often unable to reproduce the sentences from the testing set. However, in some cases, a different set of character doesn't mean that where are facing a wrong reconstruction: the model

input	*reconstruction*
It was terrible.	it is worse than the matrix sequels.
this movie was craptacular.	this movie was pure trash.
have you ever watched it?	why don' t do it?
otherwise, it did suck.	otherwise, don't bother.
the plot is almost nonexistent.	the plot makes no sense at all.
nothing could have helped.	nothing makes much sense.
first of all, the title daily is a lie.	first of all, it was a disappointment.
this film has no life in it.	this film however, is a void.
the suspense is laughable.	the story is pletty stupid.
avoid this movie it makes you bitter.	avoid this movie at all costs, folks.
where is the truth ?	why is the truth ?

Table 4.4: Hand-picked reconstructions. These example show an ability of the model to interpret unknown sentences with its state of knowledge

4.5.2.1 Fit to $\mathcal{N}(0, 1)$

After training, we obtain a latent loss of 19.7 for the training set and 19.5 for the testing set. The model performs equivalently with known and unknown data. However, this term is difficult to interpret: further investigations are required.

The figure 4.3 shows the distributions of samples z drawn from $q_\phi(z|x)$ where x is a mini-batch of 5000 data points. Each sub-graph represents an independent dimension of z. The distribution $\mathcal{N}(0, 1)$ is presented as a reference. Plots indicate a good fit to $\mathcal{N}(0, 1)$. However, it is difficult to evaluate the quality of the distribution at very small scale: overall the model seems to fit $\mathcal{N}(0, 1)$ but it can present some local distortions.

4.5.2.2 Smoothness and *density*

In order to check the smooth and *dense* nature of the latent space, we need to ensure that every point within the latent distribution $p(z) = \mathcal{N}(0, 1)$ generate valid sentences and we need verify if sentences decoded from a neighborhood are similar with regard to the meaning. Hence, naming f the generative model, we propose the following steps as evaluation technique:

- First, we sample x_0 from $p(z) = \mathcal{N}(0, 1)$
- Second, we sample neighboring points $x = x_0 + \epsilon * r$
- Third, we assess qualitatively the proximity of $f(x)$ with $f(x_0)$ with regards to the meaning

In practice, sampling directly z from $\mathcal{N}(0, 1)$ showed that the latent space is very sparse: decoded z are often grammatically incorrect and the model often fails to generate meaningful words. As a consequence, we conclude that the latent representation is not *dense*.

We can now pursue with analysis of the model and check if the model is locally continuous. For this purpose, we choose x_0 as an element of the training set. The table **??** shows three examples of neighborhoods but similar results can be observed with other points. In this case, we observe that the model fails to generate perfectly correct sentences. Nonetheless, by interpreting the noisy outputs, we observe that those neighbors seem to be related to the same concepts. For instance, the sentence *"the writing was terrible"* has *"the writing, though, was unrealistic"* as neighbor. Both sentences are about the writing and convey a negative point of view.

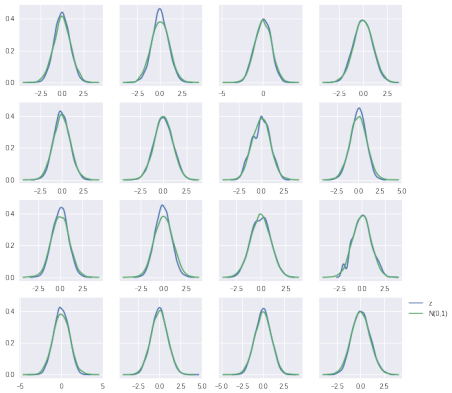


Figure 4.3: This figure shows the distribution of samples $z \sim q_\phi(z|x)$ for each dimension of the latent space. As a reminder, we assume $p(z) = N(0, 1)$ as a constraint. The distribution of latent representations seems to fit $\mathcal{N}(0, 1)$

the acting is very poor!	**the actors look as if they are drugged.**	**the writing was terrible.**
the acting is even some pretty dook.	the actor who played jesus was miscast.	the writing, these isn't really bad.
the acting is perhaps the more everynde.	the actor really gets what he paid for.	the writing, though, was unrealistic.
the acting is perhaps the more ever need	the actor who played jesus was miscast.	the writing, though, was unrealistic.
the acting is perhaps the dose even ne.	the actor who played jesus was miscast.	the writing, though, was untereating.
the acting is perhaps the dore desient.	the actor really gets what he paid for.	the writing, this were interesting.
the acting is ever some pretty dork.	the actor really gets what he paid for.	the writing, this were interesting.
the acting is perhaps the more everynde.	the actor wholl is hard wath hes belut.	the writing, though, was unterlibating.
the acting is perhaps the dose even need	the actor who played jest was mistart.	the writing, though, was unrealistic.
the acting is perhaps the vose ending.	the actor who piace ias wast has beadt.	the writing, though, was unrealistic.
the acting is even some pretty dork.	the actor really gets what he paid for.	the writing, these isn't realistic.

Table 4.5: Sentences decoded from the neighborhood of an initial point (first row). As we see, reconstructions are not perfect but the model seems to be locally continuous with regards to the meaning: neighboring sentences have similar meaning to the input sentence

4.6 Conclusion

We have now built and trained a natural language model based on the Variational Recurrent Auto-Encoder framework. In a second phase, we proposed methods to assess the performances of the model regarding two points: the quality of the reconstructions and the regularization quality with a focus on the continuous nature of the latent space which is required for generative tasks.

We observe that the model the latent representation is very sparse. Nevertheless, despite poor generative performances, the similarity of the sentences within a neighborhood and the case of the rephrasing phenomenon seems to indicate a good recognition performances and a certain softness.

CHAPTER 5

Adapting the Linguistic Style

We introduced the Variational Recurrent Auto-Encoder in the context of natural language modeling. This model, despite being trained on a very small dataset, seems to have acquired a high-level representation of the language and presents some smoothness in the language space: similar points in the latent space corresponds to sentences with similar meanings. Furthermore, we have seen that the model tend to rephrase unknown sentences using its state of knowledge. This rephrasing ability is the first step towards style transfer because generated sentences were conveying the same meaning as the input sentences but were using a different style.

Similarly to the model described in [GEB15], we aim to build a high-level representation of the language with separate components for the style and the content. Furthermore, [RJS17] has proven that neural networks can learn to model stylistic features such as the sentiment in an unsupervised fashion. Furthermore, [SHB16] has also revealed that the use of side constraints could be effective for style modeling.

First, we will inspect the latent space of our model for stylistic features such as the positivity and negativity. Then, we will propose practical methods to adapt the linguistic style using such model.

5.1 Sentiment as Stylistic Feature

5.1.1 Style and Sentiment

Style in language can take many forms. For instance, [PK99] provides a few examples including the pronounced use of long words, the use of first person singular, the use of negations and the presence of positive or negative emotions or the use of causations. Furthermore, the border between style and content is blurry because different styles often come with a different content. For instance, people using the first-person singular heavily in their conversations will use a more direct linguistic style but they will also relate things from their point of view which can distort the content. Additionally, people using positive emotions and negative emotions will tend to present a different view on the same object. As a result, we should keep in mind that a change of style will likely come with a slight change of content.

In natural language processing, sentiment refers to the attitude of a speaker with respect to some topic. For instance, in the context of the film reviews, the review of a person who disliked a movie will be labeled with a negative sentiment. By contrast, a review from a fan of the same movie will be labeled with a positive sentiment.

The sentiment lies on the limits of the definition of the linguistic style because sentences conveying very different sentiments carry different meanings. For instance, a negative movie review will say that a particular movie is bad and a good one will transmit a positive view on the same movie. Therefore it seems that modifying the style will inevitably change the content. Nonetheless, sentiment is interesting because it comes in many shades. Indeed, we can find reviews carrying positive views about the same movie which use a different degree of positivity. For instance, the sentences *"I like this movie."* and *"I love this movie."* transmit the same content but use different styles expressed as a difference of sentiment: one use more positive emotions. Thus, slight variations of sentiments can be considered as variations

of style. Furthermore, diametrically opposite sentences on the sentiment spectrum won't carry the same meaning but can address the same object. For instance, the sentence *"I loved the sequence when James Bond was riding in Istanbul"* and the sentence *"I hated the sequence with James Bond riding in Istanbul"* exhibit a very different point of view. However, both sentences inform that James Bond was riding in Istanbul. As a consequence, even if the point of view of the author is changed, we can keep an important component of the content while significantly changing the sentiment.

Last, the sentiment is a good candidate for practical reasons. First, sentiment analysis is a field widely studied and measuring tools are largely available. Second, datasets focusing on the sentiment are available. Third, the sentiment is a feature that exhibits a large panel of nuances which are interesting in the context of a qualitative study.

5.1.2 Sentiment Analysis

Sentiment Analysis refers to the use of natural language processing tools to measure the affective states of a speaker towards a particular topic. Sentiment analysis models are usually based on machine learning techniques. Typically, models are trained using a labeled dataset such as product reviews: the review itself is the object to classify and the note of the review is the label.

A wide variety of techniques from the simple frequency analysis (which words occur more frequently in the context of positive or negative reviews) to complex character-based deep learning models are used.

In the context of the project, we have chosen to use the Large Movie Review Dataset from [Maa+11] because it provides reviews with their corresponding ratings. The ratings would have been used as a sentiment metric. Unfortunately, we split the reviews into sentences because processing a whole review was computationally too intensive and because it allows us to increase the total number of examples. However, sentences themselves taken out of their context lack of coherence with the overall rating of a review. Indeed, a bad review doesn't necessarily mean that every sentence in the reviews is negative. Indeed, despite a generally negative point of view, the author may have liked some parts of the movie and have written few positive lines about it.

As a response to this problem, we use an external sentiment analysis tool to measure sentences. We chose to use the [HG14] for this project. Vader is a good compromise between availability and performances. Indeed, Vader is directly included into the NLTK, the popular open source Natural Language ToolKit widely used in the machine learning community. Furthermore, as described in the original paper (released in 2014): "Vader performed as well as (and in most cases, better than) eleven other highly regarded sentiment analysis tools" and "Vader outperforms individual human raters". Vader predicts a sentence to be of 3 different sentiments: negative, neutral or positive. Some examples are provided in the table **??**.

5.2 Unsupervised Learning and Sentiment

We now have decided to work with the sentiment as a stylistic feature to model. We will now present arguments in favor of the unsupervised learning of the linguistic style and we will present the results based on a few experiments.

5.2.1 Fundamental Motivations

In machine learning, unsupervised models are trained without any additional data but the data points themselves. In the context of this project, the model is fed with chains of integers corresponding to the encoding of the different characters. No additional information such as the sentiment value or any other label is provided to the model. Thus, it seems that there is no reason that such model could learn to model sentiment because the sentiment is a human concept.

sample	neg	neu	pos
bad, sad, and rubbish.	0.767	0.233	0.0
great old movie drama.	0.0	0.423	0.577
i always enjoy thomas gomez.	0.0	0.484	0.516
it is a perfect scene.	0.0	0.448	0.552
this film sucks!	0.583	0.417	0.0
the two actors are very natural.	0.0	0.642	0.358
it was hilarious.	0.0	0.426	0.574
it was dumb.	0.623	0.377	0.0
well acted and good enough plot.	0.0	0.444	0.556
it was weak, very very weak.	0.592	0.408	0.0
it' s horrible.	0.778	0.222	0.0
nothing more, nothing less.	0.0	1.0	0.0
she' s a good actress.	0.0	0.408	0.592

Table 5.1: Randomly selected sentences from the Large Movie Review Dataset and their associated sentiment found using the Vader sentiment analysis tool

However, the design of the model is the key motivation behind the assumption that the model can learn stylistic feature. Indeed, the model is designed on the top of the auto-encoder framework and the particularity of the Variational Auto-Encoder is to have a limited capacity due to its regularization. As a consequence, the model has to learn to encode the input sentences into an optimal code in order learn an important number of sequences. Thus, the latent representation is the optimal encoding of the language.

We will define the concept of *pure* information as the component of information which is independent of the encoding. This is the actual information we want to transmit using a media (voice) and an encoding (language). In this sense, the content of a sentence is *pure* information.

The optimal encoding contains the element of *pure* information to transmit otherwise it couldn't be decoded while preserving its content. On the other hand, the optimal encodes nothing more than the *pure* information otherwise it would see its size growing. As a result, we can argue that the optimal encoding is *pure* information.

In the context of the Variational Auto-Encoder, the latent representation is the content conveyed by the speech because the latent representation is the optimal encoding of the speech. For instance, elements in the latent space may be the information *people think that Matrix is a good movie* or *Orson Welles has a good reputation*. Then, the job of the decoder is to decode this language into natural language such as *I love Matrix.* or *Orson Welles is such a good director.*

However, the information *people think that Matrix is a good movie* can be decoded into *I love Matrix* or *I like Matrix*. Thus, the language contains more its apparent content; it contains the choice of expression which is an expression of the context. Indeed, the choice between *like* or *love* depends on the context: this is the speaker's favorite movie or maybe the speaker tends to use superlatives more frequently. Thus, the actual content is formed of the apparent content and the context and as the context is part of the content, context is *pure* information. Because linguistic style is an expression of the context, as stated in the section 1.1, then the linguistic style is pure information. Therefore the linguistic style is likely to be represented as an explicit feature in the optimal encoding of the language.

5.2.2 Sentiment Distribution and Sentiment Feature

If the model explicitly learns stylistic features of the language, then the stylistic features are discriminative features of the sentences in the latent space. As a result, sentences with different styles will be located in different areas of the latent space. On the other hand, if the style is not represented in the latent space, sentences with different styles will be distributed uniformly in the latent space.

The figure 5.1 displays the distribution of positive, neutral and negative sentences in the latent space. Each subgraph corresponds to a dimension of the latent space. Overall, sentences are uniformly distributed except for some dimensions which present different distribution for the positive, neutral and negative sentences. For instance, over dimension number 11, neutral sentences are mostly distributed on the negative values of z and other sentences are mostly distributed on the z positive values of z.

We use the KL divergence to measure the divergence between the latent positions of the mostly negative and mostly positive sentences for each dimension of the latent space. We find that the mean KL Divergence is about 0.13 and the variance is about 0.0017 which indicates a narrow distribution of divergences over the dimensions. Last, we find that one dimension which stands out of the crowd with a divergence of 0.25.

We investigated further the discriminative potential of this dimension by analyzing qualitatively examples generated using similar the same latent code except for the value of the most discriminative dimension. We report that in a few cases, sentences generated over this axis present variations in the use of the verb such as *like* or *love* but this phenomenon remains isolated and can't be considered as a general result.

5.3 Adapting the Style

The Variational Auto-Encoder has shown to be able model high-level features of the language such as style and content. We will now propose methods to adapt the linguistic style. The four methods are represented in a schematic way in the figure 5.2.

5.3.1 Dimension Feature

[RJS17] has shown the ability of a model trained in an unsupervised fashion to adapt the linguistic style. Indeed, in this paper, one neuron corresponds to the sentiment and fixing this neuron during a generative process allows us to generate reviews with a positive or a negative point of view on the same object.

However, the analysis of the sentiment with regard to each dimension in section 5.2.2 has proven the non-ability of our model to model such feature.

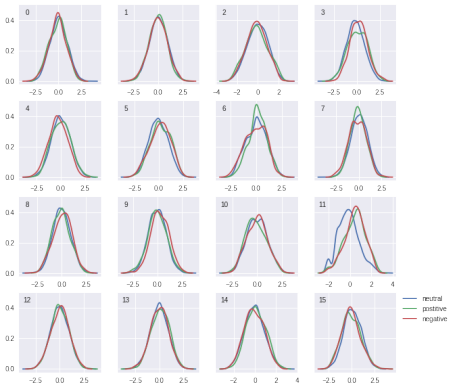
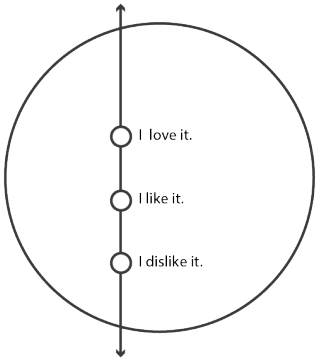
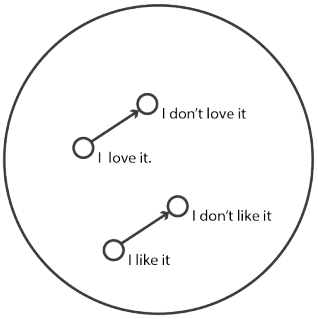


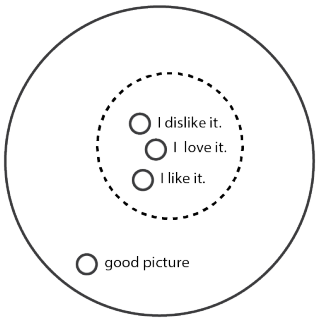
Figure 5.1: Distribution of positive, neutral and negative sentences in the latent space. Each sub-graph corresponds to a dimension of the latent space. Overall, sentences are uniformly distributed except for some dimensions which present different distribution for the positive, neutral and negative sentences



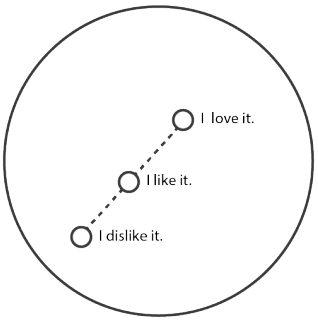
((a)) Sentiment feature



((b)) Linear Transformation



((c)) Neighborhood



((d)) Interpolation

Figure 5.2: The four different methods proposed to modify the linguistic style

5.3.2 Linear Transformations in the Latent Space

[Mik+13] introduced the Skip-gram model[1] which allows us to represent words encoded into a sparse one-hot vector into a compact representation. These word embeddings are interesting because they represent words at the concept level and this high-level space presents interesting properties. Indeed, some language patterns are explicitly represented as linear transformations in the embedding space. For instance, the result of a vector calculation vec("Madrid") - vec("Spain") + vec("France") is closer to vec("Paris") than to any other word vector. We can conclude that the relation vec([Country]) -> vec([Capital]) is explicitly represented in the structure of the embeddings and is shared among the examples of this rules such as vec("Madrid") -> vec("Spain") and vec("Paris") -> vec("France").

The Skip-gram models build embeddings using the context in which the words are used. Thus, words used in the same context will share similar embedding. Thus, words representing similar concepts are placed side-by-side in the embedding space.

Similarly to the Skip-gram algorithm, the Variational Auto-Encoder encodes sentences in a space of a lower dimensionality. Additionally, similarly to the Skip-gram embeddings, the sentences embeddings

[1] the word2vec model is an implementation of this model

tend to be related to high-level concepts because adjacent sentences in the latent spaces share similar meanings.

Linear transformations in the concept space such as observed with the word embeddings seem to be applicable to the latent representation of the sentences. Concepts such as negation seem to be encoded in a way they can be transferred to other sentences in this space. For instance, let set z_a the encoding of the sentence a *"I liked it."* and z_b the encoding of the sentence b *"I didn't like it."* Then the vector $z_b - z_a$ must represent the concept of negation because this is what differentiate the two sentences. Finally, if we encode the sentence c *"I recommend this movie."* into z_c, then the code $z_d = z_c + z_b - z_a$ is expected to be decoded into the sentence d "I don't recommend this movie." because we apply the concept of negation $z_b - z_a$ to the sentence c. In practice, we obtained the sentence *"i did not recommend this movie this one."*: a result which presents interesting similarities with the expected result.

Applying this transformation seems to be applicable to others cases despite noisy results. Some other examples are available in the table 5.2. We can see that transformed sentences are not always correct but we can observe that the model does not simply add "don't" before the verb to create a negation: it seems that the model has partially learned some basic grammar rules. For instance, *"I was"* is transformed into *"I am not"* and *"I like it very much"* is transformed into "I didn't find it very enjoyable".

This finding confirms that transfers of concepts using linear transformations as observed with the Skip-gram model are possible in the latent representation of the Variational Auto-Encoder. In the future, we hope to apply such technique to more advanced linguistic concepts and apply it to the task of the adaptation of the linguistic style.

5.3.3 Neighborhood

We observed that similar codes are decoded into similar sentences. As a consequences, sentences $f(z)$ decoded from a neighborhood of an input sentence $f(z_0)$ will be similar to the input sequence with regard to the meaning. Instead of generating sentences by controlling the a specific feature in the latent space, we propose to generate sentences within a neighborhood and select the one that present the right characteristic with regard to the style. In this way, we expect the generated sentence keeps a meaning similar to the input sentence but uses the right style.

The existence of sentences conveying similar meanings but using different styles in a neighborhood has been presented in the section 4.5.2.2.

Original Sentence	**Negation**
I recommend this movie.	i did not recommend this movie this one.
I know what this movie is about.	i don't know why this movie is hotted.
I was captivated by her.	i am not a parent, neither am i a movie.
I watched this movie two times.	i can't wait to see this movie to sinh.
i like it very much!	i didn't find it very enjoyable to it.
i think it is only available on vhs.	i didn't know anything about it either.
i like it very much!	i didn't find it very enjoyable to it.
i will definitely buy it.	i absolutely don't really like this film

Table 5.2: Some examples of linear transformation using the vector vec("I didn't liked it.") - vec("I liked it".) which seems to represent the concept of negation. Transformed sentences are not always correct but we can observe that the model does not add simply *"don't"* to create a negation. For instance, *"I was"* is transformed into *"I am not"* and *"I like it very much"* is transformed into *"I didn't find it very enjoyable"*

5.3.4 Nuances of Style: Homotopies

The Variational Auto-Encoder has shown to be able to build a "dense" and smooth high-level representations of complex data structures such as images, music and natural language. In the case of the language, [Bow+15] exhibits a very interesting property of the latent representation of the language: the existence of homotopies. In other words, the existence of homotopies makes possible to interpolate between two sentences in the language space using interpolations of their latent representations.

For instance, let set z_1 and z_2 two points in the latent spaces and $f(z_1)$ and$f(z_2)$ their respective decoded sentences. Then, it has been observed that points $z = (1-t)z_1 = tz_2, t \in [0,1]$ are decoded into correct sentences (because the latent space is continuous) and that sentences $f(z)$ are a smooth transition from $f(z_1)$ to $f(z_2)$ with regard to the meaning.

In the context of this project, this property is particularly interesting because it could be applied to control the artistic style in an analogical fashion. For instance, we could obtain a sentence slightly positive by taking an interpolation of a very positive sentence and a very negative sentence. In other words, the existence of homotopies allows us to adapt the linguistic style with nuances.

We will now investigate the compatibility of our model with the homotopic property. For this purpose, we select points within the latent space and greedily decode the intermediate points. We evaluate the quality of the results using a qualitative approach. The table 5.3 shows an example of interpolation between the sentences "I loved it." and "I hated it.". Despite having a scattered language representation (decoded sentences are not correct), the model exhibits an interesting property: the verbs used in the interpolated sentences are intermediate concepts between the initial verbs *love* and *hate*.

I loved it.	**The music was grandiose.**	**I loved the acting**
i loved it, my wife loved it.	the music and script was maghel gaod.	*i loved* khisewtere than the latter.
i loved it, my wife loved it.	the music and scrne was a *very good* mnl.	i loved thi titler watch the ending.
i loved it, my wife loved it.	the music and scrne was a very good movi	i loved ihe tiilewand and the time.
i loved the original scary movie.	the music and scone was a very good movi	i loved ih the filetlana change ahen.
i loved the film'anemation wou.	the music and songs are also will good.	i loved it better than the matrix !
i liked the firmt onder it off.	the music and songs are also well good.	i could iever watch the entire family.
i liked the oold on gatbat tovies.	the music and songs are also *well good.*	i couldn't believe what i was seeing.
i liked the ooigina swive s lot.	the music and songs are also well good.	*i did like* the rest times that rathly.
i liked the film and ihist time.	the music and songs are also will good.	i did like the rest tean in the matrrx.
i will get the point of the film.	the music and songs are also walt aodent	i did like the restaten she thanking.
i will get this doiet on testla s.	the music and songs are also walteton.	i did like the reseaten shr thenking.
i wished she told him to stuff it.	the music and songs are also waste of .	i did like the reseaten shrtt the acting
i wited the tilm and she r bit onls.	the music was awso waste offail thang.	i did like the reseaten shrat the acting
i hated this oilm lelevise though.	the music was also waste offai they.	i did like the reseaten shrat thengs.
i hated this oilm televis d though.	the music was also waste of a fan time.	i did like the reseater save that trap.
i hated this lilm teee on the tiallg	the music was also waste of a fal attet.	i did like the ceseaating a thai thete.
i hated this lilm the firmt time a.	the music was also w waste of a falm.	*i didnlike* the cried and the trinking.
i hated this lilm the trieiti night.	the music was also a waste of a must.	i didn't find the anier in the tating.
i hated this silmed the frieg ends.	the music was also a waste of time.	i disn't fied the arien entertaining.
i hated this sileed throw it blcks.	the music was also a *waste of time.*	*i disliked* the linht acting happens.
I hated this film.	**the music was just awful.**	**I disliked the acting.**

Table 5.3: Decoded sentences of the interpolations $z = (1-t)z_1 = tz_2, t \in [0,1]$ with z_1 and z_2 the codes of the sentences *"I loved it."* and *"I hated it."*. Despite having a scattered language representation (decoded sentences are not correct), the model exhibits an interesting property: the verbs used in the interpolated sentences are intermediate concepts between the initial verbs. Relevant concepts are shown in italics

5.4 Prototype: The Pessimistic Machine

In this section, we will introduce a practical implementation of the style adaptation. In the previous section, we have seen that the sentiment was not modeled as a sentiment dimension. Furthermore, the use of linear transformations in the latent space is promising but this method couldn't be applied to the sentiment feature. As a consequence, the neighborhood within the latent space seems to be the most usable method for a practical application.

Use case The aim of this prototype is to produce a conversation at cross-purposes between the user and a virtual agent for entertainment purposes. In this context, we expect the user to talk about a particular movie using short sentences such as *I liked the acting* or *the music was really nice*. In response to the comment, the Pessimistic Machine will address the same topic using a negative point of view. For instance, given the input sentence *I loved this movie*, the model is expected to answer *I hated this movie*. A mockup of the application is shown in the figure 5.3.

Method We have seen that the model was performing quite poorly at generating sentences because the latent space is very sparse. As a consequence, using the model in a full generative fashion seems impossible for practical use. However, in the section 4.5.1 we have shown that the model was good at rephrasing unknown sentences which indicates that the model features a robust recognition model. This is why we chose to construct this prototype using the recognition model $q_\phi(z|x)$ only. We will use the sentences of the training set as potential answers to ensure to produce meaningful outputs.

The prototype will work as follows. First, we need to compute the latent representation z of every sentence x using the recognition model: $z = q(x)$. On runtime, when the model receives an input sentence x_0, its latent representation $z_0 = q(x_0)$ is computed. Last, we identify potential answers by

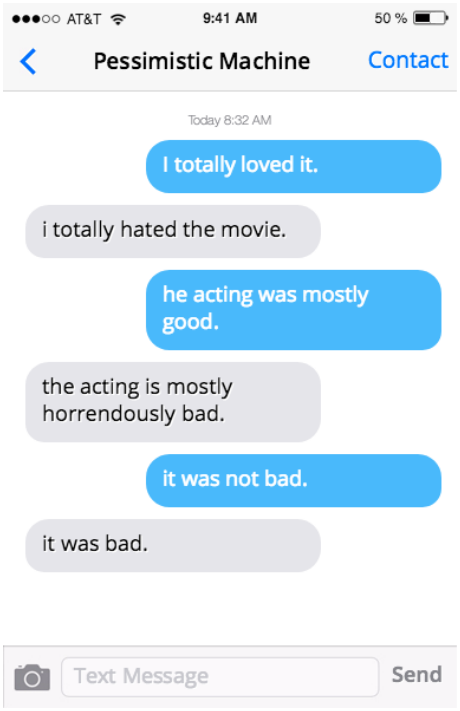


Figure 5.3: Use case of the Pessimistic Machine. The user sends messages sends a movie critic to the Pessimistic Machine answers with a similar but negative sentence. Answers are generated by the prototype

finding its k nearest neighbors from the original dataset. Finally, we identify the adequate candidate within the nearest neighbors with regard to the target linguistic style.

Evaluation Evaluating the performances of the model using a measure is it difficult because it requires using a measure which is able to catch the meaning of different sentences and compare them. For this reason, the scope of this project will be limited to the qualitative assessment only.

We will process as follows: we select a random sentence from the dataset and request an answer from the Pessimistic Machine. We also evaluate the capacity of the model answer unknown sentences by inputting them manually. A set of these answers is available in the table 5.4. Two other sets of experiments are available in the appendix D. One corresponding to samples generated from a set of human inputs and another generated from randomly selected sentences from the dataset.

5.5 Conclusion

First, we argued that the sentiment can be considered as an example of the linguistic style and presented motivations for the use of an unsupervised model to model the linguistic style. Second, we investigated the performances of the current model with regard to this task and showed that the model failed on this point. Third, we proposed four techniques to adapt the linguistic style using such representation: using an explicit stylist latent feature, transferring properties using linear transformations, using a neighborhood and finally using interpolations between sentences to adjust the style. Last we proposed an prototype which implements the style adaptation and evaluated its performances on a set of examples. We will now analyze further the results in the chapter 6.

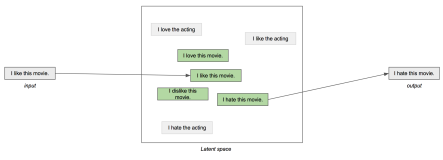


Figure 5.4: Diagram of the Pessimistic Machine. The Pessimistic machine encodes the input sentence into its latent representation. Second, sentences from the dataset are found within a neighborhood of the code of the input sentence within the latent space. Last, the sentence with the most negative sentiment within the neighborhood is returned. This sentence is expected to address the same object as the initial sentence but convey a negative sentiment

input	**answer**
I totally loved it.	i totally hated the movie.
I was really bad.	it' s really bad.
it was terrible.	thats terrible.
the acting was mostly good.	the acting is mostly horrendously bad.
I liked this movie.	i liked this movie.
it was a nice movie.	this is a waste of time and money.
the story was amazing.	the story was bad.
it was not bad.	it was bad.
The acting was good.	the irony is horrible.
The music was good.	the music is horrendous.

Table 5.4: Some answers provided by the Pessimistic Machine. The prototype handles short sentences in most of the cases. The Pessimistic Machine catches the object of the input sentence and provides an answer which addressing the same object but conveying a negative sentiment

CHAPTER 6

Discussion

We have introduced the Variational Auto-Encoder and applied the model to a natural language modeling task and investigated its use in the context linguistic style adaptation. We will now discuss the benefits and drawbacks of the choices we have made, the results of the experiments detailed in the previous sections and address the problems encountered during this project.

6.1 Variational Auto-Encoder

This project focuses strongly on the Variational Auto-Encoder which has shown to be most interesting study subject in many aspects.

First, this model makes an intensive use of neural networks. Thus, it was a unique opportunity to focus on this topic and to acquire a general knowledge in this field.

Second, this model is a complex mathematical object which pushed to gain an in-depth understanding of its underlying mathematics which helps to acquire a deeper understanding of the last advances in the machine learning community. At some extents, this knowledge gained during this work allows to become not only a user of the existing deep learning frameworks but provides with the means to be a force for bringing forward proposals.

Third, the Variational Auto-Encoder has shown to be a powerful tool in the context of building generative models for unstructured data. The model has proved to be able to model the MNIST dataset and sequential data structures such as a set of time series and, to some extent, natural language. This framework is especially interesting because it can be trained in an end-to-end fashion using the Stochastic Gradient Descent algorithm which promises that this framework can scale to more complex tasks and larger dataset. By contrast, generative model relying on sampling methods are known to scale poorly.

Last, the latent representation built using this model presents interesting properties such as the softness of the representation, a high expressivity and the existences of linear relationships between concepts.

Nevertheless, training deep and large stochastic model has proven to be a delicate task. In particular, the need to weaken the regularization loss was a key obstacle toward the obtention of better results because it drastically reduces the generative performances of the model. In this context, we suggest constructing more powerful encoder and decoder models.

6.2 Modeling Recurrent Structures

During this project, we focused on the use of the Recurrent Neural Networks as language model. As a previous step, we introduced the theory underlying the RNNs and presented a naive implementation of the model applied to the encoding and decoding of a sinusoid signal. This experiment was especially worthy because it allowed us to understand how the RNNs can transmit information over time steps.

We have seen that RNNs are relevant in the context of sequential data because they can model complex sequential behaviors. In particular, the modeling experiment of a set of sinusoids undertaken in the section 3.4 has proven that the model could encode and decode a wide variety of complex behaviors using a reasonable number of parameters.

Furthermore, coupled with LSTM cells, the RNNs have proven to be an adequate language model. Indeed, our model has been able to encode and decode a set of 34 490 sentences with an error rate of 3%. Nevertheless, the poor generalization of our model stresses the need for more efficient regularization techniques.

6.3 Difficulties with Training Deep and Large Neural Networks

Training deep neural architectures is a computationally intensive task. First experiments have proved that using CPU computing for such task was impracticable because of the slowness of the process. For this project, we have used the Amazon Web Services with free credits obtained in the context of the Educate program. Thanks to this program, we could run the experiments using an instance with one GPU. However, these resources are still insufficient compared to the resources available in a production context. Indeed, a larger scale model would require more than one GPU to be trained in a reasonable time. As a consequence, the size of the model trained during this project was bounded by the hardware and time availability.

Furthermore, training deep architectures is a difficult task whose difficulty decreases with general experience. This why the search of the adequate hyper-parameters has been a long and exhausting work which has result in an extensive investment of time. The advancement of this project has been most likely slowed down by a lack of experience with large and deep neural networks, but the many experiments have resulted in a fruitful gain of knowledge.

6.4 Variational Auto-Encoder, Natural Language and Style Adaptation

The choice of the Variational Auto-Encoder in the context of the style adaptation is a cornerstone of this project. We will now discuss the results about the natural language generation task and the style adaptation task.

6.4.1 Natural Language Modeling

Modeling natural language at the character level is a particularly challenging task because it requires to learn a vocabulary and requires to get to organize words into sentences. Furthermore, we must stress that the model is learning in an unsupervised fashion. Thus, no additional knowledge about the structure of the words and the grammatical rules is provided.

Because of computing power limitations, we have limited the scope of this project to a narrow dataset. However, we can argue that this restriction is a critical handicap toward the construction of a general language model because such model needs a huge number of observations to catch the underlying rules of the language. As the size of the training dataset decreases, the ability of the model to learn strong and general rules decreases as well.

Reproducing Sentences The model is trained by reconstructing the input into identical sentences. Thus, evaluating the model with regard to this task was required to assess the performances of the model. We have seen that the cross entropy is hard to interpret which motivated us to introduce the average misclassification error and to conduct a qualitative analysis. We observed that the model successfully learned to reproduce items for the training set (3% of misclassified characters) and found that the model performs quite poorly regarding generalization: sentences from the testing set were not reconstructed correctly at the character level (61% of misclassification). However, we observed that the model gained a remarkable ability to rephrase unknown sentences using its knowledge. A generated sentences could be composed of a different set of words, but a similar meaning was conveyed. For instance, *this film has no life in it.* is interpreted as *this film however, is a void.* This is a fascinating result because

the rephrasing is a successful adaptation of the style: the model keeps the same content (meaning) but changes the way to say it (style).

Generative Nature The goal of this project was to build a soft and *dense* representation of the language. Nevertheless, we have seen that the latent representation is still very sparse despite the regularization constraints. Indeed, sentences generated from random samples in the latent space are likely to be meaningless chains of characters. We remind that this aspect has been evaluated trough the qualitative analysis of random samples.

However, despite being very noisy, sentences selected from a neighborhood present interesting similarities when considering their meaning. More precisely, in the neighborhood of *"the acting is very poor!"*, are decoded the sentences *"he acting is even some pretty dook."* and *"the acting is perhaps the more ever need"* which share a similar syntactic structure and a common subject (*the acting*).

Nevertheless, the sparse nature of the latent space and the incorrectness of the decoded sentences prevent us from using this model in a purely generative context.

Performances and Scale of the Experiment It is necessary to put into perspective the size of the model used for this study. Indeed, the model was trained on a very narrow dataset. Only 34 490 sentences were used while neural translation models are usually trained using million of pairs of sentences. Therefore it is not surprising to see that the model generalizes very poorly and that the model struggles to build a continuous space since this the dataset by its small size presents some sparsity.

Furthermore, the recognition and generative models used for this experiment are very rudimentary. We expect stronger models to lower the difficulty in applying constraints on the latent space because of higher abstraction capabilities. As a result, we could use a more robust regularization on the latent variable which would improve the latent representation with regards to its *density*.

6.4.2 Style Modeling and Application Style Adaptation

We introduced the sentiment as a stylistic feature and stated our motives in favor of the use of an unsupervised algorithm to learn high-level features such as the sentiment. As a second step, we investigated the existence of the sentiment as an explicit latent variable. Finally, we proposed four different methods to adapt the linguistic style and introduced them using small examples. Finally, we introduced the Pessimistic Machine, an agent which answers movie critics with similar but negative critics.

Style as a Latent Variable Modeling the linguistic style as a latent variable was the initial strategy conceptualized in the aim of adaptation the style. Indeed, this variable could have been used as a parameter to adjust the style while generating sentences.

Methods for Style Adaptation We introduced four different methods compatible with a high-level language model:

1. Controlling an explicit latent feature corresponding to style Transferring concepts between sentences using linear transformations

2. Using a set of similar sentences using the neighboring sentences in the latent space and choosing the best fit

3. Interpolating interpolation in the latent space to adjust the style

Sentiment as a latent feature Our model has shown to be unable to model the style of as an explicit feature. Thus, we can exclude the use of this technique in the context of this project.

Linear Transformations Similarly to the case of the transfer of relations with the Skip-gram representation, linguistic rules can be transferred in our latent space. Indeed, we observed that the concept of negation could be represented by a vector in the latent space and that this vector could be applied to another sentence. More precisely, we defined the vector *vvt* = code(*I didn't like it*) - code(*I liked it*) and applied this vector to a few examples. Notably, applying this negation vector to the sentence *i like it very much!* produces the sentence *"i didn't find it very enjoyable to it."* and applying it to the sentence *"I was captivated by her."* returns the sentence *"i am not a parent, neither am i a movie."*. Despite noisy results, we observe that the model does not simply learn to add *"don't"* before the verb. Instead, the model has learned complex rules which are compatible with modal and non-modal verbs (*I am not, I can't* and *I don't known*. Furthermore, the transposition of the expression *"I like it"* into *"I didn't find it"* is notable because it shows the retention of informal rules. Indeed, we can argue that *"I didn't find it very enjoyable"* is a *better* negation of *"i like it very much!"* than *"I don't like it very much"*.

This observation is particularly interesting because it provides hints about how the model encode information. Indeed, the observation that the negation is encoded as a linear transformation seems to indicate that sentences are encoded as compositions of independent concept because the negation can be applied independently to different objects. As a consequence, it seems that the negation is encoded in a very explicit way.

It would be very interesting to investigate the existence of such explicit encoding for other linguistic rules in the context of a more advanced model. In the context of the negation, it is most likely that the model has learned this rule because of the abundance of parallel examples (i.e. *I like it.* and *I don't like it*). Thus, we expect to observe more examples of this phenomenon as the number of parallel examples increases. For instance, *I like it* and *I love it*) is another example of parallelism because the two verbs are used in the same contexts.

Neighborhood The analysis of the continuity of the latent space and the experiments with the Pessimistic Machine act as a basis of experiments for this method. Overall, we observed that the model tend to encode sentences with similar meanings in the same neighborhood which is also supported by the good recognition performances we have observed with the rephrasing cases. Additionally, we tend to observe that sentences addressing a similar object but conveying different sentiments are encoded into the same area of the latent space. Thus, we proposed to use sentences decoded from a neighborhood of the input sentence and pick the most relevant one.

Furthermore, it has been observed that the neighborhoods were more consistent for smaller sentences than for longer ones. However, one can observe that the difficulty in modeling sentences increases exponentially with the number of characters while the number of example for each length stays constant.

Interpolations The continuous nature of the latent representation of the language is particularly interesting when considering the analogical character of the linguistic style. This is particularly true in the case of the sentiment. A sentence can be either very positive or slightly positive. In the case of a perfect model, we expect that interpolating between diametrically opposite sentences generates sentences with intermediate styles while conserving the intent.

Additionally, the experiments tend to confirm this assumption. For instance, interpolating between the sentences *"I loved it."* and *"I hated this film."* generate sentences using the words *loved, liked, wished* and *hated*. Similarly, we have found sentences containing the expressions *very good, well good* and *waste of time* between the concepts *grandiose* and *awful*. This finding allows us to argue that the Variational Auto-Encoder has partially built a continuous and ordered representation of the sentiment.

6.5 The Pessimistic Machine

Regarding the Pessimistic Machine, we assessed the quality of the answers on two levels. First, we inspected the model's outputs by querying simple sentences inputted by hand such as *"I like this*

movie.”. Despite some errors, the model successfully catches the object of the sentence and adapts the sentiment. For instance, the model returns *”the acting is mostly horrendously bad”* for the input *”the acting was mostly good.”* and the model returns *”the story was bad.”* for the input sentence *”the story was amazing.”*. In each case, the subject *acting* or *story* is preserved.

Second, we analyzed how random samples taken from the dataset were processed and observed that this exercise has challenged the Pessimistic Machine. For instance, the sentence *”He should be a star.”* produces the answer *”he’ s at wrong.”*: an unsatisfactory result. Nevertheless, in the case of random samples, we observe that inputs sentences are much more complex than the ones inputted by hand which increases the difficulty of the task. Furthermore, the manual inputs were relating to examples which are highly represented in the dataset However, these results are not totally wrong because the model partially catches the object of some input sentences. More precisely, in the neighborhood of the sentence *”Carla finally gets laid.”*, we can find the sentences *”cheat on all fronts!”*, *”charming film but naive film.”* and *”Well and culmination fight full orgy.”*. The expressions *gets laid*, *cheat*, *”charming”* and *orgy* share the same semantic field. Similarly, the sentence *”warning this could spoil your movie.”* share a neighborhood with the sentences *”watch this movie forget your troubles.”*, *”what is wrong with some of you?”* and *”watch this at your own risk.”* which share the semantic field of the danger or risk (*warning, risk, troubles*).

This experiment is most interesting because it shows that the model has built a high-level representation of the language organized around the concepts of individual words: simple sentences with a similar object are represented in the same neighborhood and sentences sharing a same semantic field are grouped together. This result stresses further the good recognition performances of the model. Both experiments allow us to argue that the model interprets sentences at a conceptual level.

Additionally, the Pessimistic machine proves that the Variational Auto-Encoder can be used in the context of a style adaptation task. More precisely use of a strong negativity mimics a specific personality trait: being *grumpy*. Thus, we can imagine that such model could be used to create a more complex and subtle personality which would be very interesting to build human-like virtual assistants. Regarding the translation industry, this method could be used to generate different versions of the same translations and let the customer chose the one which uses the adequate linguistic style.

6.6 Difficulties in Measuring the Performances

Evaluating the performances of the model was problematic because the representation we are aiming to build is dealing with human concepts such as the intent and the linguistic style. Regarding the language modeling task, we have first introduced the cross entropy to measure the performances of the model. The cross entropy is interesting to compare different models or datasets because this shows the model measures itself its performance during the training. However, the interpretability of the metric is very limited. As a result, we introduced a human-readable metric: the average number of misclassification per character. This metric is interesting in the situation where we expect a given input to be reconstructed exactly in the same way. Nevertheless, the ability of the model to re-phrase unknown sentences shows the limit of this measure because we can argue that a sentence reconstructed with the same meaning but a different style is a good result.

Therefore, a new measure which compares the meaning of sentences would be a good asset in the context of such project because it would help to assess the performances of the model. Indeed, in the context of this project, most of the performances evaluation has been done by analyzing qualitatively random samples. Better measures would help to compare performances in a more consistent fashion and would be an important asset in order to improve the current model.

Regarding the style adaptation task, current tools such as the Vader sentiment analysis tool were sufficient to measure the change of style since we have chosen to focus on the sentiment. However, we can easily imagine that applying this method to other linguistic styles would require developing additional measuring tools.

CHAPTER 7

Future Work

During this project, we introduced the Variational Auto-Encoder and applied it to a narrow language generation task and to a simple style adaptation task. the model has shown to perform poorly on the generation task only but presents good performances on the recognition task. However, despite of the improvements introduced in the section 3.3, the model is still very rudimentary: dropout, bidirectional RNN and teacher forcing are commonplace. In this section, we will introduce a few improvements that could greatly improve the model and propose further applications.

7.1 Larger Dataset

Deep Learning algorithms are greedy, and the availability of data limits possibilities. Fortunately, the Variational Auto-Encoder is trained in an unsupervised fashion. Fortunately, the quantity of text available on the Internet is limitless. Indeed, data can be taken from many different sources such as social media, Wikipedia, forums or books.

We expect that using more data will significantly improve the performances of the model because the model will be presented with much more examples of the same grammatical rules. Indeed, understanding how to use the verb *to be* may be difficult with 1 000+ examples, however, it becomes more trivial using a million of examples.

7.2 Hyper-parameters tuning

Tuning deep learning models manually is a long and tedious task because it requires many attempts and each attempt takes a lot of time. Furthermore, manual searching is likely to provide us with a sub-optimal set of hyperparameters. However, using automated parameters search methods was not doable for this project given that it requires much more computing resources. Using better hyper-parameters tuning techniques is most likely to improve the current model.

7.3 Improving the Generative Model: Adversarial Networks

Generating realistic samples using generative models is a difficult task. In the computer vision field, generated images are often blurry. In the context of this project, we have seen that produced samples tend to be very noisy. [Goo+14] proposes a solution to improve the quality of the generated samples. The strategy employed here is to use one generative network G and one discriminative network D which recognizes *fake* samples. Both networks are trained against each other: the generator network learn to produce samples that fool the discriminator network D and the discriminator network learn to differentiate true samples from fake ones. In this way, the discriminator becomes better and better at identifying fakes items and the generator becomes better at producing fooling items. As a result, the generator becomes better at producing realistic sentences.

Such concept could be applied to the task of natural language generation and could improve the generative performances. [Wu+17] has shown that such principle could be used to improve machine translation models.

7.4 Hierarchical Encoder and Decoder

In the section 4.1.1, we discussed the advantages and disadvantages of using character-level or word-level encoding. Using a character level encoding comes with many advantages, but this encoding increases significantly the number of steps to encode with the RNN. As a result, the capacity to encode longer sentences or entire documents is bound by the vanishing gradients problem and the limited hardware capacity.

In response to this problem, we introduced a hierarchical structure which can be used for the recognition model $q_\phi(z|x)$. The solution consists in using a word model on the top of the character model. First, a character model is used to create embeddings from the chain of characters. This model is a simple bidirectional character RNN which encodes words in both positive and negative directions. The cornerstone consists in building the embeddings from the hidden state after the word for the positive direction of the RNN and preceding the word the negative direction. In this way we allow the model to create a representation of the word using both the chain of characters of the world itself and the surrounding context (previous words and future words).

The architecture is represented in the figure 7.1. Regarding this example, the embedding of the word *like* is a concatenated vector composed from the encoding of *I like* and *like trains*: both the word and its context are encoded.

This architecture is directly inspired by the Skip-gram model presented in [Mik+13]. the Skip-gram representation of words depends on the context in which the words are used. Indeed, the training objective of the Skip-gram model is to find representations of words that are useful for predicting the surrounding words in a sentence. Thus, embeddings are a context-dependent representation of words. Because of the success of the Skip-Gram model, we hope that implementing a model which can consider not only the chain of character forming the word itself but the surrounding context can help to create better embeddings.

This architecture is a solution for both the vanishing gradient problem and the computing power limits because it allows us to us a narrower at the character level. Indeed, the information doesn't need to be carried over the whole sentence. Thus, the capacity of the character RNN can be decreased which limits de facto the overall complexity of the model. Second, the information of the whole sentence will be carried over the word-level RNN which is made of a smaller number of steps.

This model is also attractive to encode longer language structures such as a whole document. Indeed, this architecture is expected to scale well because additional layers can be built. A first layer which creates word embeddings from characters, a second layer which generates sentences embeddings from word embeddings and the last layer which creates the final document embedding.

This architecture has shown to be efficient in the context of a neural Question-Answering task has experimented by Yannis Flet-Berliac in his Master Thesis.

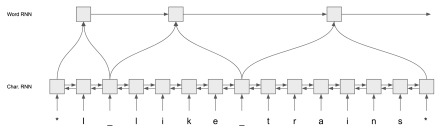


Figure 7.1: the char2word architecture allows us to takes profit from both the character-level encoding and the word-level encoding. The solution consists in creating embeddings from the characters. Embeddings are created using both the word itself and the past and previous words thanks to a bidirectional RNN at the character level. Embeddings are then fed into the word RNN

7.5 A more Expressive Latent Space: Hierarchical Latent Representation

In the context of Deep Learning models, it has been observed that deep architectures were able to model unstructured data with a high level of abstraction, each layer creating a more abstract representation from the previous layer in an iterative fashion.

More particularly, in the computer vision field, the use of a multi-layered architecture has shown to be a good solution to represent images with different levels of abstraction. For instance, in [GEB15], it ha been observed that the artistic style (or texture) corresponds to the lowest layers of the deep convolutional model and that the content corresponds to the latest layers.

Unfortunately, such hierarchical structure does not exist in the context of the Variational Recurrent Auto-Encoder: only one stochastic layer is used. However, it is possible to create a multi-layered representation of the data by using multiple stochastic layers. Unfortunately, introducing multiple layers of stochastic variable increases the difficulty in training the model. [Kaa+16] proposes a solution to this problem by constructing a model with multiple stochastic layers corrected by a term depending on the data itself. In practice, this makes possible to train deep stochastic models.

The Ladder Variational Auto-Encoder has shown to outperform the simple Variational Auto-Encoder framework by creating a more expressive latent representation of the data. We propose to investigate the use of such framework for the task of natural language generation in the context of style modeling for future work.

CHAPTER 8

Conclusion

The goal of this thesis was to build a general language model with the aim of adapting the linguistic style. This project was strongly inspired by the the artistic style transfer as it has been performed in the computer vision field. The translation industry's needs to build better natural language processing tools and the opportunity to improve current conversational interfaces by providing the tools to understand and generate natural language at a human level are the main motivations behind this project.

Driven by the last advances in natural language modeling, we proposed the use of the Variational Auto-Encoder framework for the purpose of building a high-level representation of the language with the aim of building an explicit representation of the linguistic style.

We have focused first on the theory behind the framework itself and presented a first use case: modeling the MNIST dataset. Motivated by the sequential nature of the language, we introduced the Recurrent Neural Network and demonstrated its ability to encode and generate sinusoids. Thereafter, we implemented the Variational Recurrent Auto-Encoder which uses Recurrent Neural Networks as encoder and decoder. Lastly, we validated the viability of such model by testing it against a simple study case: the modeling of a set of time series.

As a second step, we applied our model to the modeling of the natural language. In this context, two major choices had to be made. First, we chose to encode sentences at a character level. Second, we decided work with the Large Movie Reviews Dataset.

Our model has shown poor generative performances which is presumably due to the limited size of the dataset and the necessity to train the model with weak regularization constrains. Nevertheless, the model has proven to be good at recognizing the meaning in the natural language which is notably supported by its capacity to rephrase unknown sentences.

We argue that, at some extent, our model has built a high-level representation of the language at a content and style level which is supported by three notorious cases. First, the model is able to represent syntacticly and stylistically similar sentences into similar codes. Second, the model has acquired an inner representation of high-level concepts such as negation. Third, the model has proven to have built an ordered representation of concepts which is supported by the interpolation case.

We proposed methods to adapt the style and introduced a simple prototype: the Pessimistic Machine and showed that the *neighborhood* method was viable in the case of simple queries which resulted in the observation of successful style adaptation cases. In other words, the case of the Pessimistic Machine proves that adapting the linguistic style can be achieved using the Variational Auto-Encoder.

Finally, we argued that the current model can be greatly improved and we believe that the further improvements would be greatly profitable in succeeding to generalize our findings.

We have also acquired a clearer picture of the style adaptation problem and acquired a deeper understanding of the potential and limitations of the model which both extends and nuances the field of applications. While the actual task of transferring style as observed in the computer vision field seems to remain an important step ahead, the good recognition performances of our model shorten the path to the construction of human level natural language understanding tools.

APPENDIX A

VAE: modeling hand-written digits (Jupyter Notebook)

VAE-mnist-toy

March 6, 2017

1 VAE MNIST

Generating Hand-written digits

This Notebook is inspired by this Github Tutorial

1.1 Libraries, Constants and Helpers

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import tensorflow as tf
print("Tensor Flow version {}".format(tf.__version__))
from tensorflow.contrib.tensorboard.plugins import projector
import os
```

```
Tensor Flow version 1.0.0
```

In [2]:
```
from IPython import display
class updatable_display():
    """
    display a graph which is updated every iteration
    """
    def __init__(self):
        self.X = []
        self.Y = []

    def update(self, x,y):
        """
        add value and display
        """
        self.X += x
        self.Y += y
        plt.plot(self.X,self.Y, "gray")
        display.clear_output(wait=True)
        display.display(plt.gcf())
    def close(self):
        """
        close the display
        """
        display.clear_output()
```

In [3]:
```
flags = tf.app.flags
FLAGS = flags.FLAGS
```

```
        flags.DEFINE_boolean('re_train', True, 're-Train the model')
        flags.DEFINE_float('learning_rate', 0.0005, 'Initial learning rate.')
        flags.DEFINE_integer('max_steps', 200, 'Number of steps to run trainer.')
        flags.DEFINE_integer('hidden1', 600, 'Number of units in hidden layer 1.')
        flags.DEFINE_integer('hidden2', 601, 'Number of units in hidden layer 2.')
        flags.DEFINE_integer('z_dimension', 2, 'Dimension of the latent space')
        flags.DEFINE_integer('batch_size', 128, 'Batch size.')
        flags.DEFINE_string('train_dir', 'mnist_train_200iters', 'Directory to put the training data.')
        flags.DEFINE_string( 'log_dir', "tensorboard/mnist", "directory for variable summaries, made to

        print "tensorboard --logdir="+ FLAGS.log_dir
```

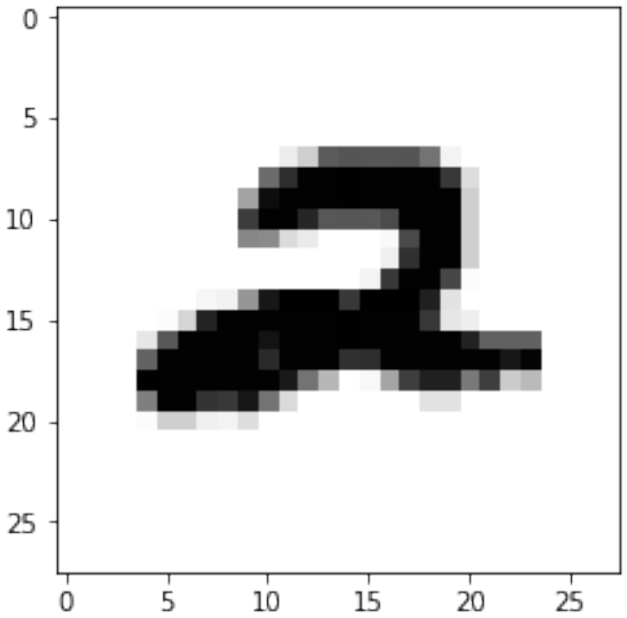
```
tensorboard --logdir=tensorboard/mnist
```

2 Load dataset

```
In [4]: from tensorflow.examples.tutorials.mnist import input_data
        mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
        n_samples = mnist.train.num_examples
        print("Number of samples {} Shape of y[{}] Shape of X[{}]"
              .format(n_samples, mnist.train.labels.shape, mnist.train.images.shape))
        plt.imshow(np.reshape(-mnist.train.images[4242], (28, 28)), interpolation='none',cmap=plt.get_cm
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Number of samples 55000 Shape of y[(55000, 10)] Shape of X[(55000, 784)]
```

```
Out[4]: <matplotlib.image.AxesImage at 0x7fdef6b4abd0>
```



3 TensorFlow

3.1 Helpers

In [5]:
```
def variable_summaries(var):
    """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
    with tf.name_scope('summaries'):
        mean = tf.reduce_mean(var)
        tf.summary.scalar('mean', mean)
        with tf.name_scope('stddev'):
            stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    #    tf.summary.scalar('stddev', stddev)
    #    tf.summary.scalar('max', tf.reduce_max(var))
    #    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)

def weight_variable(shape):
    """
    define a weight variable
    """
    return tf.Variable(tf.truncated_normal(shape, stddev=0.1))


def bias_variable(shape):
    """
    define a bias variable
    """
    return tf.Variable(tf.constant(0.1, shape = shape))

def nn_layer(input_tensor, input_dim, output_dim, layer_name, act=tf.nn.relu):
    """Reusable code for making a simple neural net layer.

    It does a matrix multiply, bias add, and then uses relu to nonlinearize.
    It also sets up name scoping so that the resultant graph is easy to read,
    and adds a number of summary ops.
    """
    # Adding a name scope ensures logical grouping of the layers in the graph.
    with tf.name_scope(layer_name):
        # This Variable will hold the state of the weights for the layer
        with tf.name_scope('weights'):
            weights = weight_variable([input_dim, output_dim])
            #variable_summaries(weights)
        with tf.name_scope('biases'):
            biases = bias_variable([output_dim])
            #variable_summaries(biases)
        with tf.name_scope('Wx_plus_b'):
            preactivate = tf.matmul(input_tensor, weights) + biases
            #tf.summary.histogram('pre_activations', preactivate)
        activations = act(preactivate, name='activation')
        #tf.summary.histogram('activations', activations)
        return activations

def distribution_feature( input_tensor, input_dim, output_dim, layer_name ):
    """
```

```
        define a feature which represents part of a distribution.
        For example, to represent Gaussian distribution, this layer can represent mu or log(sigma^2)
        """
        with tf.name_scope(layer_name):
            with tf.name_scope('weights'):
                weights = weight_variable([input_dim, output_dim])
                #variable_summaries(weights)
            with tf.name_scope('biases'):
                biases = bias_variable([output_dim])
                #variable_summaries(biases)
            with tf.name_scope(layer_name):
                feature = tf.add(tf.matmul(input_tensor,weights), biases)
                variable_summaries(feature)
            return feature
```

3.2 Encoder

```
In [6]: # Input
        n_input = 28*28
        n_labels = 10
        n_hidden_1 = FLAGS.hidden1
        n_hidden_2 = FLAGS.hidden2
        n_z = FLAGS.z_dimension

        x_input = tf.placeholder("float", shape=[None, 28*28], name="x_input") #Batchsize x Number of P
        #y_ = tf.placeholder("float", shape=[None, 10])   #Batchsize x 10 (one hot encoded)

        with tf.name_scope("Encoder"):
            q_h_1 = nn_layer(x_input,n_input, n_hidden_1, "q_layer_1", act = tf.nn.softplus)
            # Second hidden layer
            q_h_2 = nn_layer(q_h_1,n_hidden_1, n_hidden_2, "q_layer_2", act = tf.nn.softplus)

        # Parameters for the Gaussian
        with tf.name_scope("z"):
            # parameters of the gaussian (distribution of the latent space)
            # the distribution z is therefore parametered by the mean mu and the variance sigma^2
            z_mu = distribution_feature(q_h_2,n_hidden_2,n_z, "mu" )
            # a little trick from the original tutorial: sigma > 0 but we don't want to enforce the net
            # therefore we ask the network to model the parameter log(sigma^2) in [-inf,+inf]
            z_ls2 = distribution_feature(q_h_2,n_hidden_2,n_z, "log_sigma_power2" )

        #z_mean = tf.add(tf.matmul(q_h_2, weights([n_hidden_2, n_z])), bias([n_z]))
        #z_log_sigma_sq = tf.add(tf.matmul(h_2, weights([n_hidden_2, n_z])), bias([n_z]))
```

3.3 Decoder

```
In [7]: batch_size = FLAGS.batch_size
        with tf.name_scope("Decoder"):
            with tf.name_scope('random_sample'):
                eps = tf.random_normal((batch_size, n_z), 0, 1, dtype=tf.float32) # draw a random numbe
            with tf.name_scope('z_sample'):
                z = tf.add(z_mu, tf.multiply(tf.sqrt(tf.exp(z_ls2)), eps))  # a sample it from Z -> z
                variable_summaries(z)
```

```
            # first hidden layer
            n_hidden_1 = FLAGS.hidden1
            p_h_1 = nn_layer(z,n_z, n_hidden_1, "p_layer_1", act = tf.nn.softplus)
            # second hidden layer
            n_hidden_2 = FLAGS.hidden2
            p_h_2 = nn_layer(p_h_1,n_hidden_1, n_hidden_2, "p_layer_2", act = tf.nn.softplus)

        # reconstruction distribution
        with tf.name_scope("x_reconstruction_mean"):
            x_reconstr_mean = tf.nn.sigmoid(tf.add(tf.matmul(p_h_2,  weight_variable([n_hidden_2, n_inpu
            #x_mu = distribution_feature(p_h_2,n_hidden_2,n_input, "mu" )
            #x_ls2 = distribution_feature(p_h_2,n_hidden_2,n_input, "log_sigma_power2" )
```

3.4 Loss function

```
In [8]: with tf.name_scope("Cost"):
            with tf.name_scope('reconstruction_loss'):
                reconstr_loss = -tf.reduce_sum(x_input * tf.log(tf.clip_by_value(x_reconstr_mean,1e-10,
                                               + (1-x_input) * tf.log(tf.clip_by_value(1 - x_reconstr_m
                variable_summaries(reconstr_loss)
            with tf.name_scope('latent_loss'):
                latent_loss = -0.5 * tf.reduce_sum(1 + z_ls2 - tf.square(z_mu) - tf.exp(z_ls2), 1)
                variable_summaries(latent_loss)
            with tf.name_scope('cost'):
                cost = tf.reduce_mean(reconstr_loss + latent_loss)   # average over batch
                variable_summaries(cost)

        with tf.name_scope('train'):
            optimizer =  tf.train.AdamOptimizer(learning_rate=FLAGS.learning_rate).minimize(cost)
```

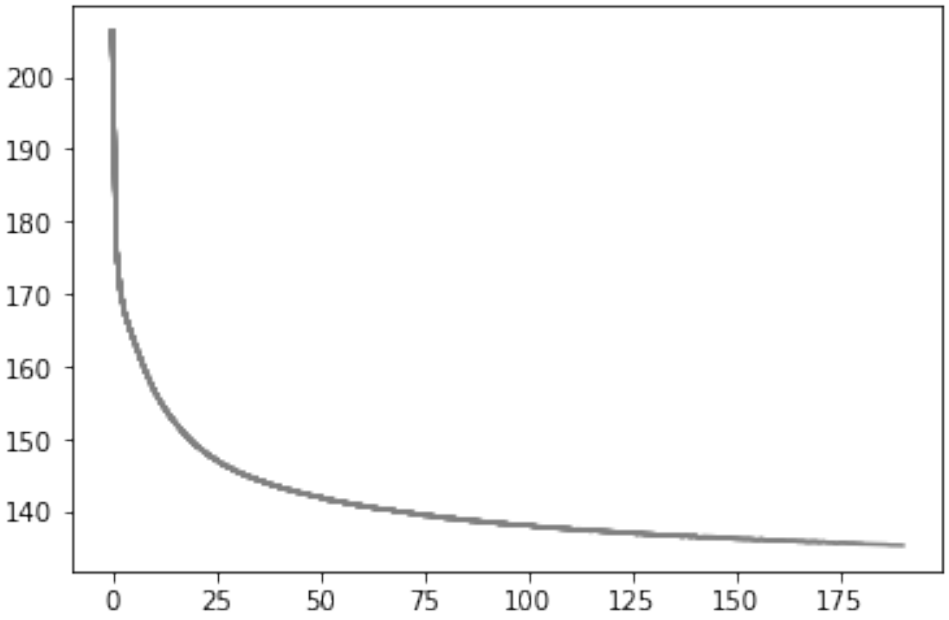
3.4.1 summaries

```
In [16]: sess = tf.Session()
         merged = tf.summary.merge_all()
         train_writer = tf.summary.FileWriter(FLAGS.log_dir,
                                              sess.graph)
```

4 Training

```
In [10]: if FLAGS.re_train:
             runs = FLAGS.max_steps #Set to 0, for no training
         else:
             runs = 0
         init = tf.global_variables_initializer()
         saving_path = FLAGS.log_dir
         saver = tf.train.Saver()

         dis = updatable_display()
         with tf.Session() as sess:
             sess.run(init)
             # TensorBoard
             writer = tf.summary.FileWriter(saving_path)#, graph=tf.get_default_graph())

             batch_xs, _ = mnist.train.next_batch(batch_size)
```

```
print(batch_xs.shape)
dd = sess.run([cost], feed_dict={x_input: batch_xs})
print('Test run after starting {}'.format(dd))

xx = []
yy = []

for epoch in range(runs):
    avg_cost = 0.
    total_batch = int(n_samples / batch_size)
    # Loop over all batches

    for i in range(total_batch):
        batch_xs, _ = mnist.train.next_batch(batch_size)
        _,d, summary = sess.run([optimizer, cost , merged], feed_dict={x_input: batch_xs})
        avg_cost += d / n_samples * batch_size
        # write log
        writer.add_summary(summary, epoch * total_batch + i)

    xx.append(epoch)
    yy.append(avg_cost)
    # Display logs per epoch step
    if epoch % 10 == 0:
        save_path = saver.save(sess, saving_path ) #Saves the weights (not the graph)
        #print("Model saved in file: {}".format(save_path))
        #print "Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost)
        dis.update(xx,yy)
        xx = []
        yy = []
dis.close()
```

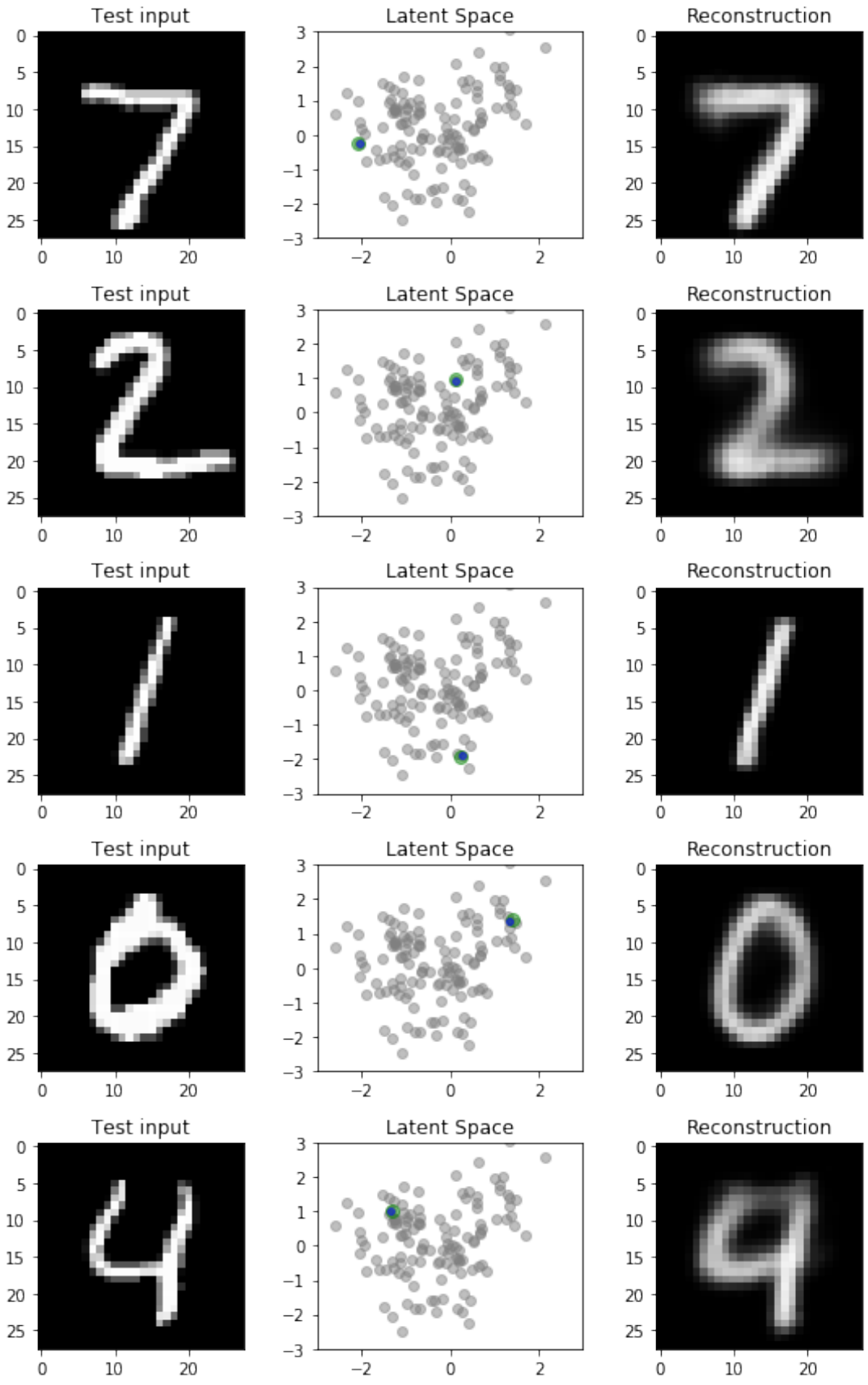


5 Tests

5.1 reconstruction of a real hand-written digit

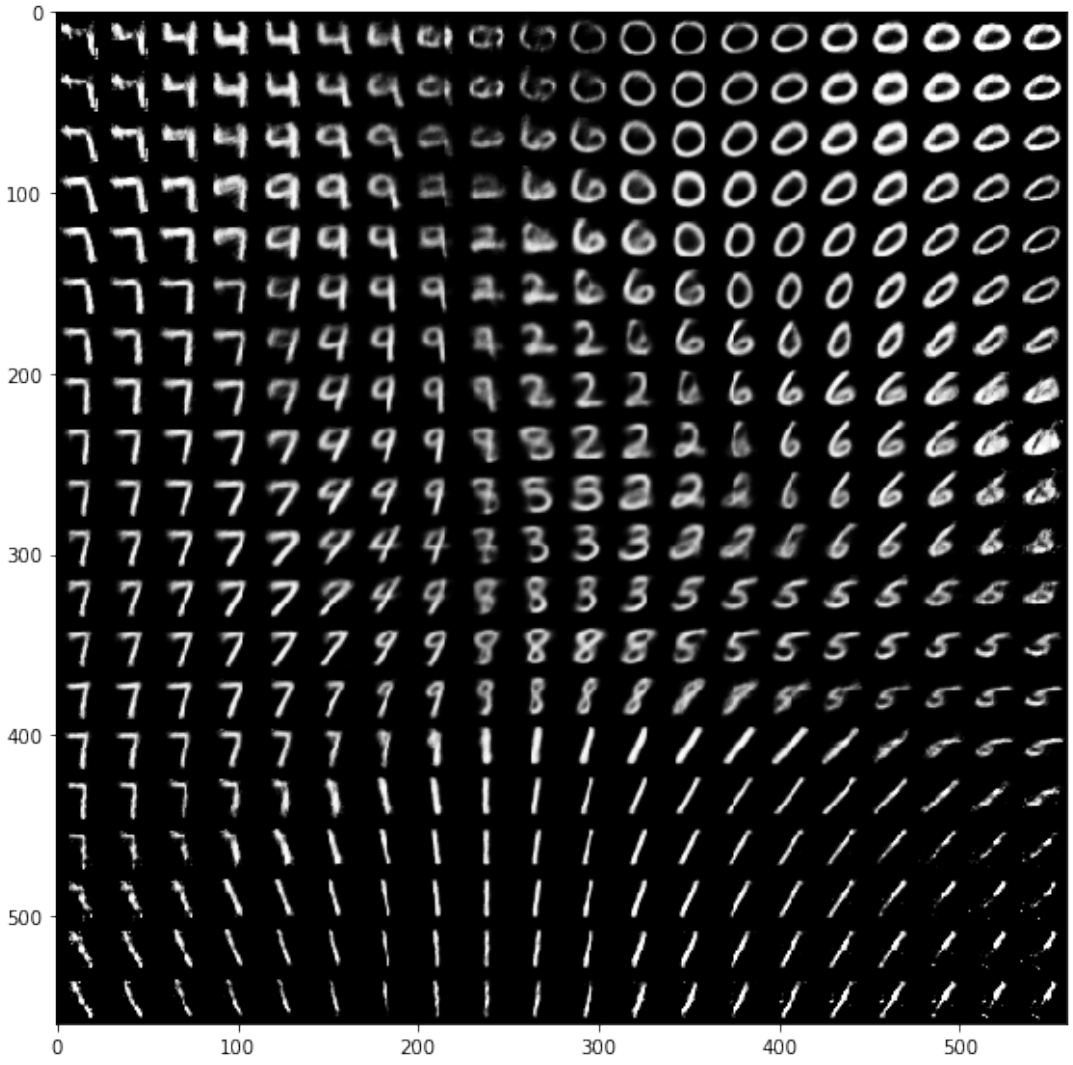
In [11]:
```
saver = tf.train.Saver()
print saving_path
with tf.Session() as sess:
    saver.restore(sess, "./"+saving_path)
    print("Model restored.")
    x_sample = mnist.test.next_batch(batch_size)[0]
    x_reconstruct,z_vals,z_mean_val,z_log_sigma_sq_val  = sess.run((x_reconstr_mean,z, z_mu, z

    plt.figure(figsize=(8, 12))
    for i in range(5):
        plt.subplot(5, 3, 3*i + 1)
        plt.imshow(x_sample[i].reshape(28, 28), vmin=0, vmax=1,  interpolation='none',cmap=plt
        plt.title("Test input")

        #plt.colorbar()
        plt.subplot(5, 3, 3*i + 2)
        plt.scatter(z_vals[:,0],z_vals[:,1], c='gray', alpha=0.5)
        plt.scatter(z_mean_val[i,0],z_mean_val[i,1], c='green', s=64, alpha=0.5)
        plt.scatter(z_vals[i,0],z_vals[i,1], c='blue', s=16, alpha=0.5)

        plt.xlim((-3,3))
        plt.ylim((-3,3))
        plt.title("Latent Space")

        plt.subplot(5, 3, 3*i + 3)
        plt.imshow(x_reconstruct[i].reshape(28, 28), vmin=0, vmax=1, interpolation='none',cmap
        plt.title("Reconstruction")
        #plt.colorbar()
    plt.tight_layout()
```

```
tensorboard/mnist
Model restored.
```


Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction

5.2 Latent Space

In [14]:
```
nx = ny = 20
x_values = np.linspace(-3, 3, nx)
y_values = np.linspace(-3, 3, ny)
canvas = np.empty((28*ny, 28*nx))
saver = tf.train.Saver()

with tf.Session() as sess:
    saver.restore(sess, "./"+saving_path)
    for i, yi in enumerate(x_values):
        for j, xi in enumerate(y_values):
            z_mu = np.array([[xi, yi]])
            d[0] = z_mu
            x_mean = sess.run(x_reconstr_mean, feed_dict={z: d})
            canvas[(nx-i-1)*28:(nx-i)*28, j*28:(j+1)*28] = x_mean[0].reshape(28, 28)

plt.figure(figsize=(8, 10))
Xi, Yi = np.meshgrid(x_values, y_values)
plt.imshow(canvas, origin="upper", vmin=0, vmax=1,interpolation='none',cmap=plt.get_cmap('gray
plt.tight_layout()
```



In [ ]:

APPENDIX B

VRAE: modeling timeseries (Jupyter Notebook)

VRAE+Timeseries

May 11, 2017

1 VRAE: timeseries

This notebook showcases the use of VRAE to learn a ensemble of time series and generate them from a latent contiunous variable. The exercice is motivated by the desire to build this model on a simple example (sinusoids) and adapt it afterwards to more complicated sequential data: natural language.

The ensemble of functions is a group of sinusoids generated by two uniform independent variables which modulate the phase and the frequency of the curves.

The implementation has been inspired by this implementation of the VRAE by the author with Thenao, the seq2seq tutorial and an implementation of a VAE with Tensorflow.

And RNN tutorials this and this

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        import tensorflow as tf
        print("Tensor Flow version {}".format(tf.__version__))
        from sklearn.preprocessing import OneHotEncoder
        from IPython import display
        import time
        import pickle

        class updatable_display():
            """
            display a graph which is updated every iteration
            """
            def __init__(self, headers=None):
                if headers != None:
                    self.X = []
                    d = dict()
                    for h in headers:
                        d[h] = []
                    self.ys = d

            def update(self, x,ys):
                """
                add value
                """
                self.X += x
                for h in self.ys.keys():
                    self.ys[h] += ys[h]
            def display(self,keys= None,live = True):
                """
                plot the training data
```

```
        """
        if keys == None:
            keys = self.ys.keys()
        plt.clf()
        fig, axes = plt.subplots(1, len(keys), figsize=(len(keys) * 5,5))
        fig_i = 0
        for h in keys:
            ax1 = axes.flatten()[fig_i]
            ax1.plot(self.X,self.ys[h],"gray")
            ax1.set_title(h + ": " + str(self.ys[h][-1])[:6])
            #ax1.annotate(self.ys[h][-1],xy=(   , np.mean(self.ys[h]) ) )
            fig_i+=1
        fig.tight_layout()
        if live:
            display.clear_output(wait=True)
            display.display(plt.gcf())
            plt.close()
        else:
            plt.plot()
            plt.show()

    def close(self):
        """
        close the display
        """
        display.clear_output()

    def save(self, filename):
        """
        save data to file
        """
        save = dict()
        save["x"] = self.X
        save["ys"] = self.ys
        pickle.dump( save, open( filename, "wb" ) )

    def load(self,filename):
        "load class from file"
        save = pickle.load( open( filename, "rb" ) )
        self.X = save["x"]
        self.ys = save["ys"]
```

```
/Users/valentin/anaconda/lib/python2.7/site-packages/matplotlib/__init__.py:1035: UserWarning: Duplicate
  (fname, cnt))

Tensor Flow version 1.0.0
```

2 Constants

```
In [2]: maxTime = 1
        num_steps = 40 * maxTime     # length of each record during training
        batch_size = 256
        state_size = 60              # RNN Cells state size
        transition_layers = []       # add fully connected layers between Decoder and stochastic layer
```

```
learning_rate = 0.0005
epoches = 5000
beta_decay_offset = 300       # Beta start to increase after this number of epoches
beta_decay_period = 1000       # Beta needs this time to vary from 0 to 1
latent_loss_weight = 0.03     # latent loss weight (from original VRAE: #Use a very wide prior t
dtype=tf.float32

data_dim = 1                  # dimension of the data: one dimension

latent_dim = 2                # latent space dimension: here we choose 2 since
summaries_dir = "VRAE"
saving_path = "VRAE"
```

3 Deterministic Warm-up

Training VAE is a difficult task because of the complexity of the model and the presence of two terms in the loss function. For information, we want to maximize the loss function:

$$L = log(p(x))$$

However, we can show the existence of a lower bound L_v such as:

$$L \geq L_v = -D_{KL}(q(z|x^{(i)})||p(z)) + E_{q(z|x^{(i)})}(log(p(x^{(i)}|z))$$

We will call:

- the latent loss L_l:
$$L_l = -D_{KL}(q(z|x^{(i)})||p(z))$$
- the reconstruction loss L_r:
$$L_r = E_{q(z|x^{(i)})}(log(p(x^{(i)}|z))$$

Which gives:

$$L \geq L_v = L_l + L_r$$

The reconstruction loss L_r evaluates the reconstruction quality of the input.

The latent loss L_l however acts as a regularization term and ensures that prior distributions are shaped as Gaussian distribution. In practical terms, it distributes all the input into the Gaussian distribution such as our whole dataset occupy the whole latent space. For each input, this ensures that this one will be represented by a gaussian distribution in the latent space. As we force the inputs to be not a single point int the latent space but a distribution with a non null width and because our model tries to occupy the whole space, the model will place similar inputs side by side such as part their distribution are joint.

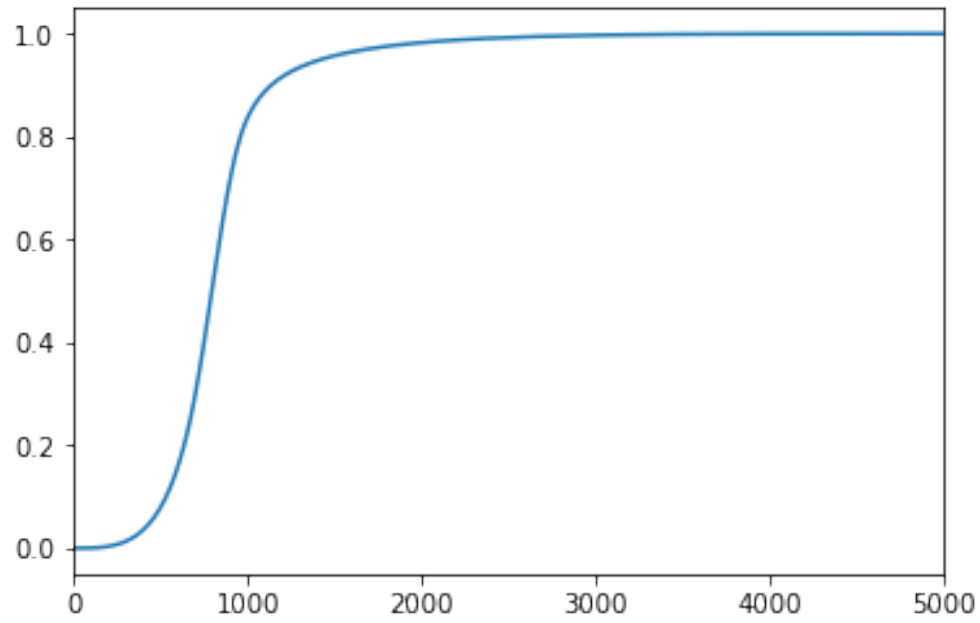
However the problem occurs when it's easier for the model to optimizes the latent loss at the expenses of the reconstruction loss. Then the model fails to find a reasonable minima. Two solutions help us to solve this problem:

- Weakening the Latent loss by simply adding a weight to this loss. This weight will be called α_l and noted "latent_loss_weight" in the code.
- Deterministic warm-up which has been proposed by this paper helps the model to focus on reconstruction during the early steps of training by starting with a null latent loss and adding progressively this term over time. The weight controlling this deterministic warm-up will be called β and noted B in the code.

Using these twos techniques we optimize:

$$L'_v = \beta\alpha_l L_l + L_r, \beta \leq 1, \alpha_l \leq 1$$

In [3]:
```
# this function creates generator which controls the value of B over time
# the generator uses B-splines in order to ensures a continuty for both the function and its de
# copied from stackoverflow
import scipy.interpolate as si
from scipy import interpolate
points = [[0,0], [0, beta_decay_offset],[0, beta_decay_offset + 0.33 * beta_decay_period], [1, 1
points = np.array(points)
x = points[:,0]
y = points[:,1]
t = range(len(points))
ipl_t = np.linspace(0.0, len(points) - 1, 100)
x_tup = si.splrep(t, x, k=3)
y_tup = si.splrep(t, y, k=3)
x_list = list(x_tup)
xl = x.tolist()
x_list[1] = xl + [0.0, 0.0, 0.0, 0.0]
y_list = list(y_tup)
yl = y.tolist()
y_list[1] = yl + [0.0, 0.0, 0.0, 0.0]
x_i = si.splev(ipl_t, x_list)
y_i = si.splev(ipl_t, y_list)
beta_generator = interpolate.interp1d(y_i, x_i)

xx = xrange(epoches)
plt.plot([beta_generator(i) for i in xx])
plt.xlim([0,epoches])
plt.show()
```



3.1 Data

Timeseries are sequence of values over time. A classical use case is the variation of price over time: at each time step is recorded value of the price of goods such as gas or a currency. Then analyst try to forecast future value in order to take better business decisions. This kind of data is difficult to model because it depends on many external factor such as social or political events or even the fluctuation of the value of other goods.

For this experiment we generate a simpler form of time series: sinusoids. Sinusoids are well fitted to test the use Recurrent Neural Networks with the VAE framework because at each step the next value can be easily inferred knowing the previous values. As we are working with a generative model, we wish not only to model a single sinusoid but a family F of sinusoids with different phases and frequencies. In order to do so, the functions $f \in F$ depend on two continuous variables x and y:

$$f : t \in \mathbb{R} \rightarrow sin(\pi x t + \pi y))$$

Finally we can define the ensemble of functions F:

$$F = \{t \rightarrow sin(\pi x t + \pi y), (x, y) \in [-1, 1]^2\}$$

The next step is to generate timeseries from the family of functions F. For the sake of simplicity, we use a fixed time vector named t. Thus, we finally have a family of timeseries S_t defined as:

$$S_t = \{f(t), f \in F\}$$

Finally, the enemble of observations in S_t are generated using two uniform variables x and y.

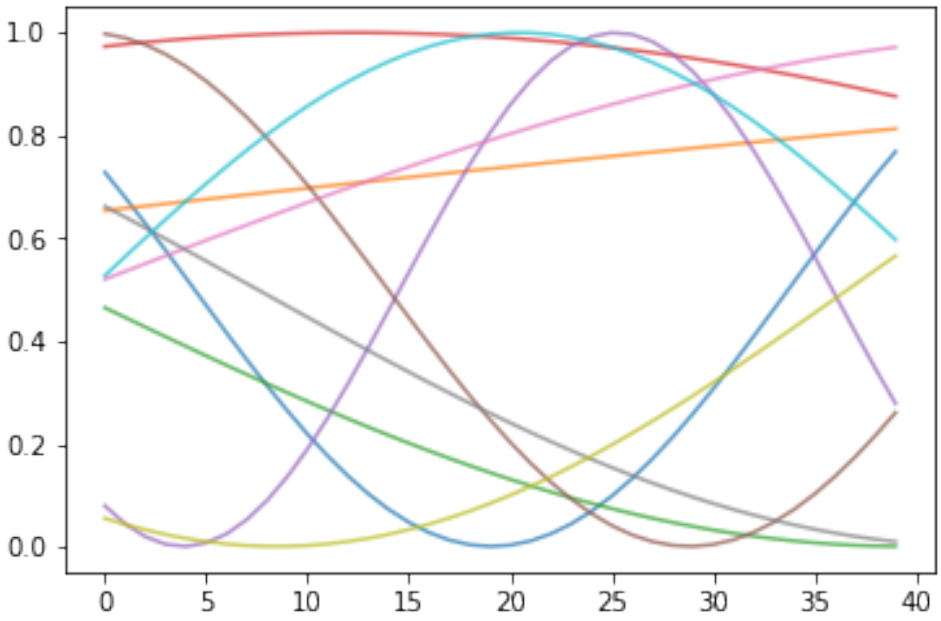
```
In [4]: t = np.linspace(-1*maxTime,1*maxTime,num_steps)    # time value for the generated sinusoids
        print "length of the time vector", len(t)

        def gen_sinusoid((x,y)):
                """
                generate samples from the latent variable (tuple x,y)
                """
                return np.array( map(lambda u :   0.5 + 0.499 *np.sin(3.14 *u * x  + 3.14 * y), t) )



        def next_batch(batch_size):
            """ Get a number of sample from our distribution of functions.

            Generate a mixture of sinusoids from a random uniform variable

            Arg:
                batch_size: (Integer) number of samples

            Return:
                a list of size batch_size of data points representing mixtures of sinusoids
            """
            # generate the prior variable
            z_1 = np.random.uniform(-1,1,batch_size)
            z_2 = np.random.uniform(-1,1,batch_size)

            return map( gen_sinusoid, zip(z_1,z_2) )
```

```
length of the time vector 40
```

3.1.1 Plot samples

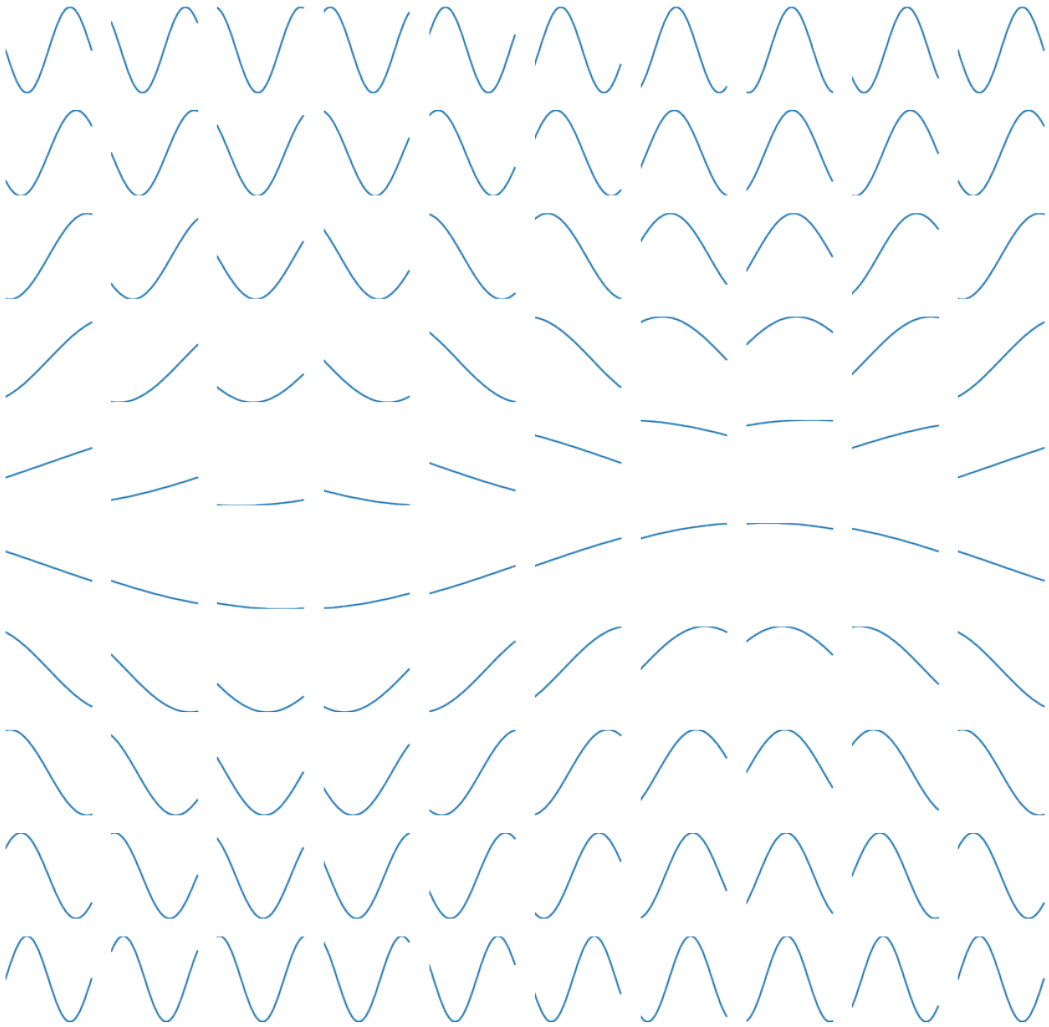
Randomly selected samples

```
In [5]: sins = next_batch(10)
        for s in sins:
            plt.plot(s, alpha=0.7)
        plt.show()
```



Samples generated from (x,y)

```
In [6]: fig = plt.figure()
        fig.set_figheight(15)
        fig.set_figwidth(15)
        #plt.tight_layout()
        i = 1
        n = 10
        for xx in np.linspace(-1,1,n):
            for yy in np.linspace(-1,1,n):
                #print xx,
                #print yy
                X = gen_sinusoid((xx,yy))
                ax1 = fig.add_subplot(n,n,i)
                ax1.plot(X)
                ax1.set_xlim([0,num_steps])
                ax1.set_ylim([0,1])
                #ax1.axis('equal')
                ax1.axis('off')
                i+=1
        plt.show()
```



3.2 The model

3.2.1 Variational Recurrent AutoEncoder

This model is an extension of the VAE with RNNs cells. It has been proposed in this paper to model and generate video games melodies.

As for the traditional VAE, This model tries to find a relation $P(X|Z)$ between the observation X and a latent variable Z. However in this case, elements drawn from X are sequences of elements like sound waves or sentences.

This model implements the encoding function $q_\phi(x|z)$ and the decoding function $p_\theta(x|z)$ as Recurrent Neural Networks. LSTM cells will be choosen because of their capacity to handle the vanishing gradients problems that occur with long term dependencies.

3.2.2 Helper Functions

```
In [7]: # prepare Tensorboard
        def variable_summaries(var):
```

```
    """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
    with tf.name_scope('summaries'):
        mean = tf.reduce_mean(var)
        tf.summary.scalar('mean', mean)
        with tf.name_scope('stddev'):
            stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    #    tf.summary.scalar('stddev', stddev)
    #    tf.summary.scalar('max', tf.reduce_max(var))
    #    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)

def weight_variable(shape):
    """
    define a weight variable
    """
    return tf.Variable(tf.truncated_normal(shape, stddev=0.1))


def bias_variable(shape):
    """
    define a bias variable
    """
    return tf.Variable(tf.constant(0.1, shape = shape))

def simple_weight_layer(input_tensor, input_dim, output_dim, layer_name):
    """
    define a layer as a simple set of weights
    """
    with tf.name_scope(layer_name):
        with tf.name_scope('weights'):
            weights = weight_variable([input_dim, output_dim])
        return tf.matmul(input_tensor, weights)

def nn_layer(input_tensor, input_dim, output_dim, layer_name, act=tf.nn.relu):
    """Reusable code for making a simple neural net layer.

    It does a matrix multiply, bias add, and then uses relu to nonlinearize.
    It also sets up name scoping so that the resultant graph is easy to read,
    and adds a number of summary ops.
    """
    # Adding a name scope ensures logical grouping of the layers in the graph.
    with tf.name_scope(layer_name):
        # This Variable will hold the state of the weights for the layer
        with tf.name_scope('weights'):
            weights = weight_variable([input_dim, output_dim])
            #variable_summaries(weights)
        with tf.name_scope('biases'):
            biases = bias_variable([output_dim])
            #variable_summaries(biases)
        with tf.name_scope('Wx_plus_b'):
            preactivate = tf.matmul(input_tensor, weights) + biases
            #tf.summary.histogram('pre_activations', preactivate)
        activations = act(preactivate, name='activation')
        #tf.summary.histogram('activations', activations)
```

```
        return activations

    def linear_layer( input_tensor, input_dim, output_dim, layer_name ):
        """
        define a feature which represents part of a distribution.
        For example, to represent Gaussian distribution, this layer can represent mu or log(sigma^2
        """
        with tf.name_scope(layer_name):
            with tf.name_scope('weights'):
                weights = weight_variable([input_dim, output_dim])
                variable_summaries(weights)
            with tf.name_scope('biases'):
                biases = bias_variable([output_dim])
                variable_summaries(biases)
            with tf.name_scope(layer_name):
                feature = tf.add(tf.matmul(input_tensor,weights), biases)
                variable_summaries(feature)
            return feature

    # from http://ruishu.io/2016/12/27/batchnorm/
    def dense(x, size, scope):
        return tf.contrib.layers.fully_connected(x, size,
                                                 activation_fn=None,
                                                 scope=scope)
    def dense_batch_relu(x, num_out, phase, scope):
        with tf.variable_scope(scope):
            h1 = tf.contrib.layers.fully_connected(x, num_out,
                                                   activation_fn=None,
                                                   scope='dense')
            h2 = tf.contrib.layers.batch_norm(h1,
                                              center=True, scale=True,
                                              is_training=phase,
                                              scope='batch_norm')
            return tf.nn.relu(h2, 'relu')
```

3.2.3 The encoder network $q_\phi(x|z)$

In [8]:
```
tf.reset_default_graph()

"""
Placeholders

each record is num_steps long
"""
x_input = tf.placeholder(tf.float32, [None, num_steps], name='input_placeholder')

#training_phase = tf.placeholder(tf.bool, name='training_phase_state')
#beta = tf.placeholder(tf.float32, name='beta_value')

"""
RNN input

The RNN API requires an additional dimension fore the dimension of each element in each record
"""
```

```
        #print rnn_inputs.shape
        with tf.name_scope("Preprocessing"):
            rnn_inputs = tf.expand_dims(x_input,-1)
            print rnn_inputs.shape
            rnn_inputs = tf.reverse(rnn_inputs, [1])

        """
        Encoder
        """
        with tf.name_scope("Encoder"):
            with tf.variable_scope('lstm_enc'):
                cell = tf.contrib.rnn.LSTMCell(state_size,state_is_tuple=True)
                #cell = tf.contrib.rnn.LayerNormBasicLSTMCell(state_size)
                #cell = tf.contrib.rnn.BasicRNNCell(state_size)
            rnn_outputs, final_state_tuple = tf.nn.dynamic_rnn(cell, rnn_inputs, dtype=tf.float32, scop
            print rnn_outputs.shape
            final_state = final_state_tuple[0] # get real state and not the hidden state (for lstm)
            tf.squeeze(final_state, name="squeeze_operation") # drop dimensions of size 1 # for lstm
            print final_state.shape
            # reduce size and batch normalization
            # encoder_out = nn_layer(final_state , state_size , decoder_out_size , "encoder_out_nn")
            #dense_batch_relu(final_state, decoder_out_size, training_phase, "encoder_batch_norm")
            if len(transition_layers) == 0:
                enc_out = final_state
                enc_size_out = state_size
            else:
                l_in = final_state
                size_in = state_size
                for s in transition_layers:
                    l_in = nn_layer(l_in, size_in, s, "enc_out_" + str(s) , act=tf.nn.relu)
                    size_in = s
                enc_out = l_in
                enc_size_out = s
```

```
(?, 40, 1)
(?, 40, 60)
(?, 60)
```

3.2.4 The latent space

```
In [9]: """
        paramterization trick
        """
        with tf.name_scope("Z"):
            z_mu = linear_layer( enc_out, enc_size_out, latent_dim, "z_mu" )
            z_ls2 = linear_layer( enc_out, enc_size_out, latent_dim, "z_ls2" )
```

3.2.5 The Decoder Network $p_\theta(x|z)$

Z sampling

```
In [10]: """
         draw a random number from z
         """
```

```
        with tf.name_scope("Z_samples"):
            with tf.name_scope('random_sample'):
                eps = tf.random_normal((batch_size, latent_dim), 0, 1, dtype=tf.float32) # draw a rand
            with tf.name_scope('z_sample'):
                z = tf.add(z_mu, tf.multiply(tf.sqrt(tf.exp(z_ls2)), eps))  # a sample it from Z -> z

        #z = tf.expand_dims(z,-1)
        print z.shape
```

```
(256, 2)
```

Output Projection Thanks ematvey for the example

```
In [11]: W_proj = tf.Variable(tf.random_uniform([state_size, data_dim], 0, 1), dtype=tf.float32)
         b_proj = tf.Variable(tf.zeros([data_dim]), dtype=tf.float32)
```

Dynamic RNN Decoder

```
In [12]: def decoder_rnn( cell_dec, z_input ):
             def loop_fn(time, cell_output, cell_state, loop_state):
                 emit_output = cell_output  # == None for time == 0
                 prev_out = cell_output
                 elements_finished = (time >= num_steps)
                 finished = tf.reduce_all(elements_finished)
                 if cell_output is None:  # time == 0
                     next_cell_state = cell.zero_state(batch_size, dtype)
                     next_input_value = tf.concat([z_input, tf.zeros([batch_size,data_dim], dtype=dtype
                 else:
                     next_cell_state = cell_state
                     next_input_value = tf.cond( #
                         finished,
                         lambda:tf.concat([ tf.zeros([batch_size,state_size], dtype=dtype),  tf.add(tf.
                         lambda:tf.concat([z_input, tf.add(tf.matmul(prev_out, W_proj), b_proj) ], 1) )
                 next_input = tf.cond(
                     finished,
                     lambda: tf.zeros([batch_size, data_dim + state_size], dtype=dtype),
                     lambda: next_input_value )
                 next_loop_state = None
                 return (elements_finished, next_input, next_cell_state,
                         emit_output, next_loop_state)
             return tf.nn.raw_rnn(cell_dec, loop_fn)

         """
         Decoder network
         """
         with tf.name_scope("Decoder"):
             # connect z to the RNN
             h_z2dec = nn_layer(z, 2, state_size, "z2initial_decoder_state", act=tf.nn.tanh)
             # RNN Cell
             with tf.variable_scope('lstm_dec'):
                 dec_cell = tf.contrib.rnn.LSTMCell(state_size)
             outputs_ta, final_state, _ = decoder_rnn(dec_cell, h_z2dec )
             # this implementation is ready for dynamic rnn
             rnn_outputs_decoder = outputs_ta.stack() # stack outputs
             print "rnn_outputs_decoder", rnn_outputs_decoder.shape
```

```
            decoder_max_steps, decoder_batch_size, decoder_dim = tf.unstack(tf.shape(rnn_outputs_decod
            decoder_outputs_flat = tf.reshape(rnn_outputs_decoder, (-1, state_size)) # flatten (batch_
            print "decoder_outputs_flat", decoder_outputs_flat.shape
            decoder_logits_flat = tf.add(tf.matmul(decoder_outputs_flat, W_proj), b_proj) # projection
            rnn_outputs_decoder = tf.transpose( tf.reshape(decoder_logits_flat, (decoder_max_steps, ba
            print "rnn_outputs_decoder", rnn_outputs_decoder.shape


        """
        Reconstruct Variables from series
        """
        with tf.name_scope("x_reconstruction"):
            rnn_outputs_decoder = tf.squeeze( rnn_outputs_decoder, [2])
            print "rnn_outputs_decoder", rnn_outputs_decoder.shape
            x_reconstr_mean = linear_layer(rnn_outputs_decoder, num_steps, num_steps, "x_mu" )
            #x_ls2 = linear_layer(rnn_outputs_decoder_stacked, num_steps, num_steps, "x_ls2" )
```

```
rnn_outputs_decoder (?, 256, 60)
decoder_outputs_flat (?, 60)
rnn_outputs_decoder (256, ?, 1)
rnn_outputs_decoder (256, ?)
```

3.3 Loss Function

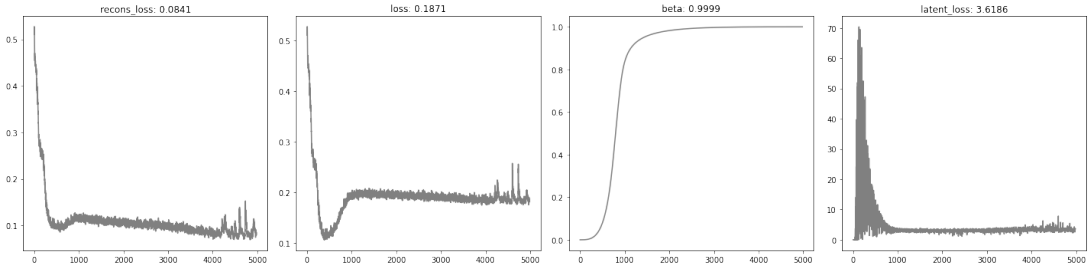
In [13]:
```
        """
        Cost: lower bound

        """
        with tf.name_scope("Cost"):
            B = tf.placeholder(tf.float32, name='B')
            with tf.name_scope('reconstruction_loss'):
                #reconstr_loss = tf.reduce_sum(0.5 * x_ls2 + (tf.square(x_input-x_reconstr_mean)/(2.0
                clip_x_input = tf.clip_by_value(x_input, 1e-8, 1-1e-8)
                clip_x_reconstr_mean = tf.clip_by_value(x_reconstr_mean, 1e-8, 1-1e-8)
                reconstr_loss = tf.sqrt( tf.reduce_mean( tf.pow( clip_x_input - clip_x_reconstr_mean, 
                variable_summaries(reconstr_loss)
            with tf.name_scope('latent_loss'):
                latent_loss = -0.5 * tf.reduce_sum(1 + z_ls2 - tf.square(z_mu) - tf.exp(z_ls2), 1)
                variable_summaries(latent_loss)

            with tf.name_scope('LB'):
                cost = tf.reduce_mean(reconstr_loss + B * latent_loss_weight * latent_loss)   # averag
                variable_summaries(cost)

        # Use ADAM optimizer
        with tf.variable_scope(tf.get_variable_scope(), reuse=False):
            with tf.name_scope('train_step'):
                optimizer =  tf.train.AdamOptimizer(learning_rate).minimize(cost)
```

4 Training

In [16]:
```
        init = tf.global_variables_initializer()
        saver = tf.train.Saver()
        # monitor training
```

```
        bs =[]
        lls = []
        rls = []
        ls = []
        dis = updatable_display( ["loss", "latent_loss" , "recons_loss" , "beta" ] )
        # training
        try:
            with tf.Session() as sess:
                start = time.time()
                tf.set_random_seed(42)
                sess.run(init)
                xx = []
                for epoch in range(epoches):
                    beta = beta_generator(epoch)
                    batch_xs = next_batch(batch_size)
                    _,d,r_c,l_c = sess.run([optimizer, cost, reconstr_loss,latent_loss ],
                                           feed_dict={x_input: batch_xs, B:[beta]})
                    xx.append(epoch)
                    lls.append(l_c[0])
                    rls.append(r_c)
                    ls.append(d)
                    bs.append(beta)
                    # Display logs per epoch step
                    if epoch % 20 == 0:
                        save_path = saver.save(sess, saving_path ) #Saves the weights (not the graph)
                        dis.update(xx,{"loss" : ls, "latent_loss":lls , "recons_loss":rls , "beta":bs}
                        dis.display()
                        xx = []
                        lls = []
                        rls = []
                        ls = []
                        bs = []
                dis.close()

        except KeyboardInterrupt:
            print('training interrupted')
        dis.display(live=False)
        end = time.time()
        print(end - start)
        dis.save('training_records/latest.p')
```

```
<matplotlib.figure.Figure at 0x7f48d16d9b90>
```



```
670.799466133
```

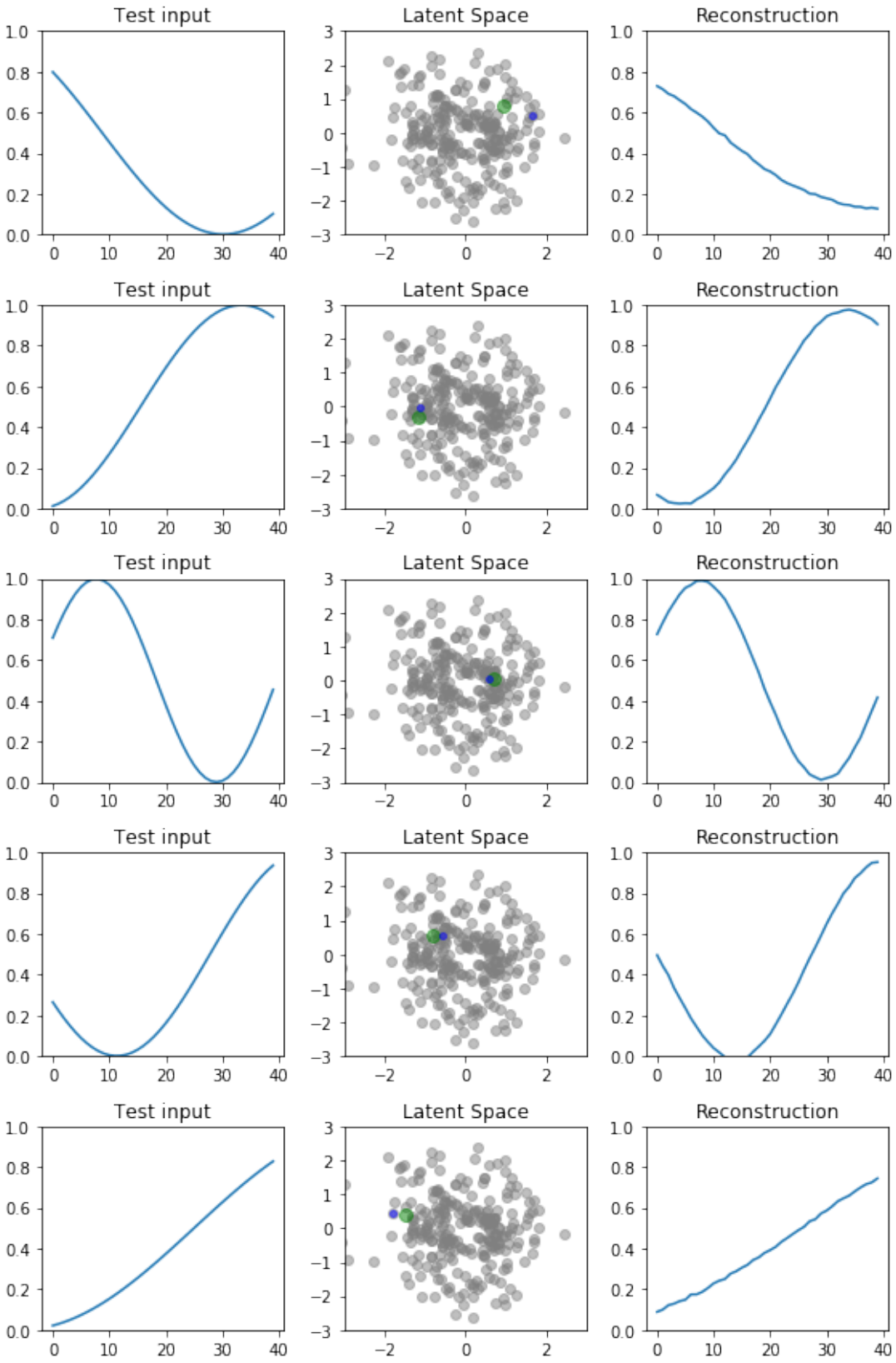
```
In [17]: print "cost: " + str(d)
         print "reconstruction loss: " + str(r_c)
         print "latent loss: " + str(l_c[-1])
```

```
cost: 0.184534
reconstruction loss: 0.0824755
latent loss: 3.83601
```

5 Reconstruction

```
In [21]: saver = tf.train.Saver()
         print saving_path
         with tf.Session() as sess:
             saver.restore(sess, "./"+saving_path)
             print("Model restored.")
             x_sample = next_batch(batch_size)
             x_reconstruct,z_vals,z_mean_val,z_log_sigma_sq_val  = sess.run((x_reconstr_mean,z, z_mu, z
             plt.figure(figsize=(8, 12))
             for i in range(5):
                 plt.subplot(5, 3, 3*i + 1)
                 plt.ylim((0,1))
                 plt.plot(x_sample[i])
                 plt.title("Test input")

                 #plt.colorbar()
                 plt.subplot(5, 3, 3*i + 2)
                 plt.scatter(z_vals[:,0],z_vals[:,1], c='gray', alpha=0.5)
                 plt.scatter(z_mean_val[i,0],z_mean_val[i,1], c='green', s=64, alpha=0.5)
                 plt.scatter(z_vals[i,0],z_vals[i,1], c='blue', s=16, alpha=0.5)

                 plt.xlim((-3,3))
                 plt.ylim((-3,3))
                 plt.title("Latent Space")

                 plt.subplot(5, 3, 3*i + 3)
                 plt.ylim((0,1))
                 plt.plot(x_reconstruct[i])
                 plt.title("Reconstruction")
                 #plt.colorbar()
             plt.tight_layout()
```

```
VRAE
INFO:tensorflow:Restoring parameters from ./VRAE
Model restored.
```

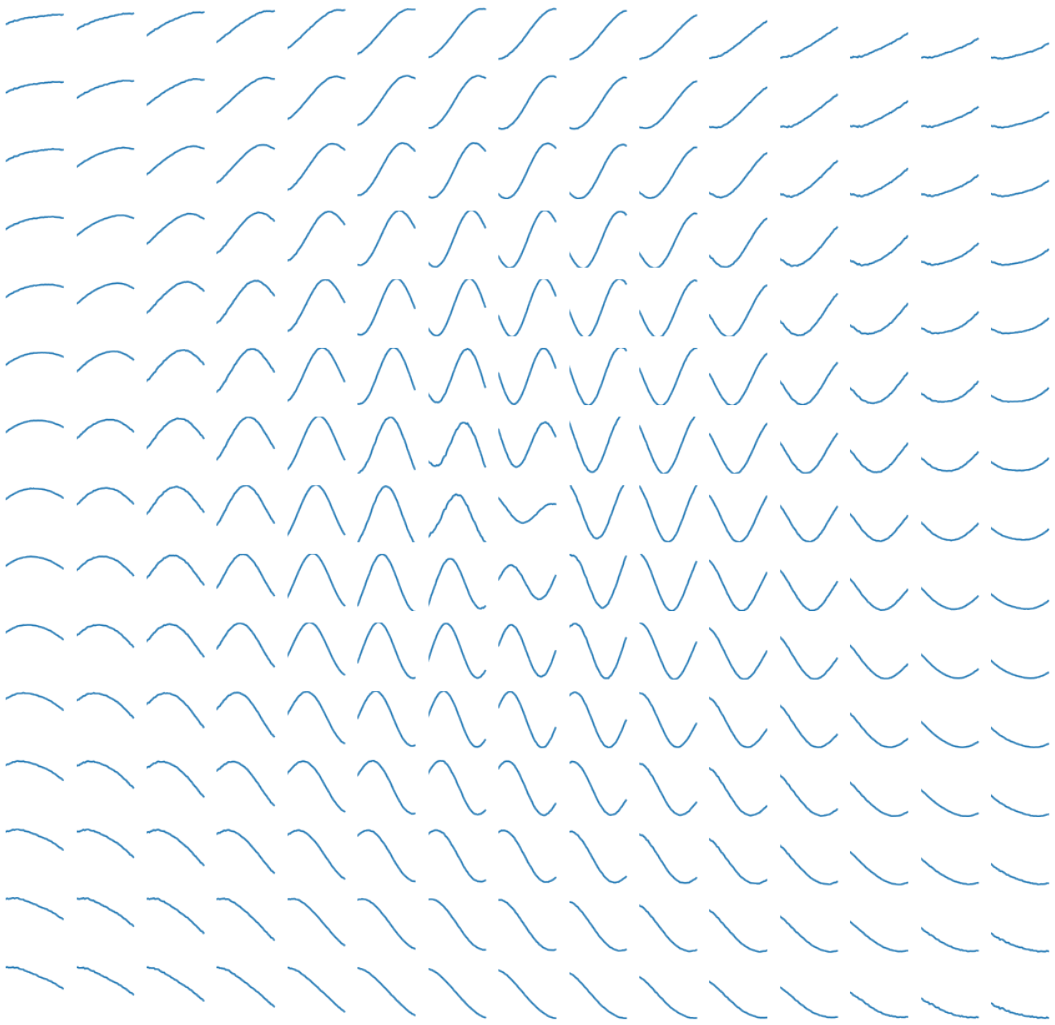

Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction
Test input
Latent Space
Reconstruction

6 Latent Space

The following plot represent the latent representation of the set of observation by the model. We sample $(z_1, z_2) \in [-1.5, 1.5]^2$ from the latent space and draw their reconstruction. Each point (x,y) in the grid corresponds to a different combination (z_1, z_2).

```
In [19]: fig = plt.figure()
         fig.set_figheight(15)
         fig.set_figwidth(15)
         #plt.tight_layout()
         n = 15
         fig_i = 1
         r = 1.5
         x_values = np.linspace(-r, r, n)
         y_values = np.linspace(-r, r, n)
         saver = tf.train.Saver()
         with tf.Session() as sess:
             saver.restore(sess, "./"+saving_path)
             for i, xi in enumerate(x_values):
                 for j, yi in enumerate(y_values):
                     z_mu_sample = np.array([[xi, yi]])
                     dd = z_mu_sample
                     ddd = [ dd[0] for x in xrange(batch_size)]
                     x_mean = sess.run(x_reconstr_mean, feed_dict={z: ddd})
                     # X = np.var(x_mean, 0) # check variance
                     X = x_mean[0,:]
                     #X = movingaverage(X, 5)
                     ax1 = fig.add_subplot(n,n,fig_i)
                     ax1.plot(X)
                     ax1.set_xlim([0,num_steps])
                     ax1.set_ylim([0,1])
                     #ax1.axis('equal')
                     ax1.axis('off')
                     fig_i+=1

         plt.show()
```

```
INFO:tensorflow:Restoring parameters from ./VRAE
```



6.0.1 Interpretation

Given a fixed time vector t, the model has succesfully learnt the space of observations with an acceptable reconstruction error (0.08). However the most remarkable result here is the ability of the model to create a latent representation of this set of observations. While the model was generated by controlling x the phase and y, here the two dimensions of z seem to control different features. This can be explained by the fact that the model doesn't learn the whole sinus function but a few data points corresponding to the time vector.

Last but not least, this example is a working example of the application of the Variation Recurrent Autoencoder to model time series. We will now apply this model to more complex data structures such as natural language.

In [ ]:

APPENDIX C

Reconstruction of Random Samples

input	**reconstruction**
you get what you pay for.	you get what you pay for.
lolita did it.	lordi i laughi
make no mistake, this is crap.	maceruwaisis not the cherism.
i never saw something like that.	i never tire of watching this.
throw it out!	throwg it wrong
save your money.	save your money.
why is that?	why is that?
out of	out of
tu pa tam is for me very bad movie.	tr il to the problbmawis af a comeny
one thing is for sure.	one think is sure too.
that makes no sense at all.	that makes no sense at all.
specialeffects?	special effects?
a gay teacher?	a gead weppen?
reba sucks.	read suuff.
the songs are instantly forgettable.	the songs are sweet and memorable.
go right ahead.	go aid that.
and the jokes aren't even funny!	and the e nonsteven gen funness
don't waste your time i wish i hadn't.	do not waste your time watching this!
man, this movie is just plain bad.	man, this movie is a buteboeton.
cheney is a very stagey early talkie.	cheese scenes are truly wilsed etcel.
i mean, horrible.	i mean, celerablo.
the lesser the better.	the ole thates perfect.
so, nothing to blow up.	so, don't listen to them
they call this film euro trash horror.	they launt mist the surp of his lock.
it is utter garbage.	it is so talknt aad.
i can't think of any anyway.	i can't wait into have anyway
please do not buy this movie!	please do not see this movie!
one of the worst movies i've ever seen.	one of the worst movies i've ever seen.
quite corny.	quite taste.
nothing comes of this development.	nothing makes to sense in her povi.
yes, the whole movie.	yes, the pool teres.
he seems to be always sermonizing.	he seems home ao an evie eyperate.
i'm not making that up.	i'm not lying about it.
im sorry to myself, you know why.	i'm sorry john, you should have wo
i'm american.	i'm married.
johansson is for once tolerable i.	john certain shoild aomos to rocess.
why watch this movie?	why was this movie d
how was your summer?	how about m nos woro?
is robocop machine or man?	is martifor hommelt ned red
nothing happened.	nothing happens.
i even liked the romance.	i remember the early days.
i couldn't enjoy the film at all.	i couldn't enjoy the film at all.
it was sooooooo long.	it was soooo ooo!
this one however is beyond bad.	this one had le rdrider cooe out
seriously, what where they thinking?	seriously what were they thinking?
who wants to be in it?	who wants to live it?

Table C.1: Randomly selected reconstructions from the VRAE model with the original input. Most interestingly, the model is often unable to reproduce the sentences from the testing set but in some cases, the model re-phrase the input sentences: outputs are different sentences but have very similar meanings

APPENDIX D

The Pessimistic Machine

D.1 Selected Samples

	I totally loved it.	I was really bad.	it was terrible.	the acting was mostly good.	I liked this movie.	it was a nice movie.	the story was amazing.	it was not bad.	The acting was good.	The music was good.
0	i totally hated the movie.	it' s really bad.	thats terrible.	the acting is mostly horrendously bad.	i liked this movie.	this is a waste of time and money.	the story was bad.	it was bad.	the irony is horrible.	the music is horrendous.
1	i absolutely loved it.	it was retarded.	it' s terrible.	the acting sucked.	i liked this movie.	it is a waste of time and money.	the story is a crap.	it was bad.	theacting was horrible.	the music was bad.
2	i absolutely loved it.	i was really lost.	it' s terrible.	the action is horrible.	i liked this movie.	it was a movie.	the storyline was a mess.	it was bad.	theacting was horrible.	the music was bad.
3	i absolutely loved it.	it is really bad.	it' s terrible.	the acting was poor.	i liked this movie.	what is a movie?	the story was all screwed up.	it was just bad.	the acting was horrific!	the cast is boring.
4	i truly laughed.	it was really really bad.	it' s terrible.	the acting was poor.	i liked this movie.	this is a tedious movie.	the story is tired and played.	it was just bad.	tragic and touching.	the case was boring.
5	i thoroughly enjoyed it.	it was really really bad.	it' s terrible.	the acting is poor.	i liked this movie.	it wasn' t a movie.	the score was amazing.	it was boring.	the acting was horrendous!	this movie was pain for me.
6	i absolutely loved this movie.	it was pretty bad.	it' s terrible.	the actors did not help matters.	i liked this movie.	i had an open mind.	the story amazing.	it was boring.	the acting was horrible.	the movie was just not good.
7	i absolutely loved this movie.	it was pretty bad.	it' s terrible.	the acting is mostly poor.	i liked this movie.	it has none.	the two aunts are a delight.	that is not bad.	the acting was horrible.	the music is good.
8	i absolutely loved this movie.	i was really bummed.	it was terrible!	the acting is horrible, save kirk.	i liked this movie.	it was a nice, subtle moment.	the story was average.	it was not.	the acting was horrible.	the music is good.
9	i absolutely loved this movie.	i' m really caught here.	it was terrible.	the characters are not compelling.	i liked this movies.	it was a good movie.	the spots were amazing.	it was not.	the acting was horrible.	the music is good.
10	i absolutely loved this movie.	it was really good.	it was terrible.	the acting was ok.	i did like this movie.	this is a vestron movie.	the story was abysmal.	it was not.	the acting was horrible.	the music was too intrusive.
11	it totally deserves more!	ah really?	it was terrible.	the attempts at humor are not.	i like this movie.	that' s a nice view.	that show was magic.	it was not.	the acting was horrible.	the music was great.
12	i absolutely love it.	i was blown away.	it was terrible.	the acting was soso.	i like this movie.	i watch it almost every day.	the cast was amazing.	thank you glend	the acting is horrid.	the music is cool enough.
13	not bloody likely.	i was blown away.	it was terrible.	the acting was only but regular.	i loved this movie.	it was made in provo 0.	the story was brilliant.	it was okay.	the acting, however, is bad.	the cast is good.
14	i just totally loved it!	i was blown away.	this was terrible.	the acting is borderline comical.	i loved this movie.	that is an incorrect view.	the soundtrack was hip, and eclectic.	it was okay.	third it was boring.	the cast is good.
15	i truly enjoyed it.	it was really incredible!	it is terrible.	the acting' s rock bottom.	i loved this movie.	with a nine mil.	the soundtrack was excellent.	it was okay.	the acting was good and bad.	the movie was good.
16	i absolutely loved it!	i warmly recommend i	it is terrible.	that was an old poem.	i loved this movie.	it has a tvmovie feeling to it.	the truth was unreal.	it was okay.	the acting was ok.	the movie was absolute garbage.
17	i absolutely loved it!	i actually feel happy.	it was too real.	the cast is good looking.	i loved this movie.	i saw it on video.	the two are wellpaired.	i want to be a friend?	the acting is wooden at best.	this movie was o.
18	i truly do.	i really laughed.	it' s actually the second.	the script are you kidding my?	i loved this movie.	it hadn' t improved much.	this was total carnage.	i think you can.	the acting is wooden and ham.	this man is a psychologist.
19		a guy fell apart.		the acting however is brilliant.		this is a decent movie.	the story is absorbing.	to anyone.	the writing, however.	the music was very good.

D.2 Randomly Selected Samples

	carla finally gets laid.	he is gunna be big in hollywood.	this is one of them.	warning this could spoil your movie.	plot twists?	this film shows a pirate hero a little!	yikes!	disappointed.	i rate this film capital f minus.	he should be a star.
0	cheated on all fronts!	his fear was because of his guilt.	this is one of them.	guess what it took to kill killjoy?	plain stupid!	this film is so bad that its painful.	yikes!	disappointed.	i hate this film with a vengsance.	he' s flat wrong.
1	charming film but naive film.	he is neither a good man or a bad man.	this is one of them.	watch this movie forget your troubles.	plain stupidity?	this movie was horrible, simply put.	yikes!	disappoint.	although i missed the last episode,	he should be ashamed.
2	well and culmination fight full orgy.	he is driven to find and stop the killer.	this is one of them.	what is wrong with some of you?	the plot was stupid.	this movie is also very sad at times.	yikes!	disappointment.	a cry in the dark is likewise clinical.	we should be ashamed.
3	walker warns pilot about sheba.	he is uproarious, daring, and moronic.	this is one of them.	watch this at your own risk.	the plot is stupid.	this film shook me when i first saw it.	yikes!	disappointing.	i rate this movie a very fair 00.	she looks up in fear.
4	randall and fischer?	he had no getup and go.	this is one of them.	watch this film at your own risk!	the plot is stupid.	this film has other flaws as well.	yikes!	disappointing.	i rate this film a 000.	he should not be trusted.
5	raul julia as general santa anna.	he is a spoiled brat and it bothers me.	this is one of them.	watch this with your soul mate.	the plot is stupid.	this film should be rated t for tripe.	yikes!	disappointing.	i rate this film a 000.	he looks old and lethargic.
6	was below standards, even for a fan.	he does great job as riddick.	this is one of them.	raisuli what kind of rifle does he use?	pretty stupid, eh?	this film came right on time actually.	yikes!	disappointing.	i rate this film a 000.	he looks about thirty.
7	wang lee hom is absolutely great.	it has dialog with bite in it.	this is one of them.	random shots of dolls.	flip on the lights?	this film should have made them stars.	yikes!	disappointing.	with all the ingredients.	who is julia to her?
8	carole landis fares best.	he' s an adult going to high school.	this is one of them.	wait till it comes out on cable.	pwog fits the bill.	this film looks so fresh and smart.	yikes!	disappointed!	i saw this title again on shemaroo.	he looks around.
9	secluded and peaceful.	so he drives off a building?	this is not one of them.	fans will spend tons of money on it.	blessings to all.	this movie was absolutely ghastly!	yikes!	disturbing.	i watched this film version of r.	he looks around.
10	hardly anyone talks.	it' d rule and you know it.	this is not one of them.	watch it with someone you truly love.	please rent this film.	this film shows all of that and more.	yikes!	delicious.	all the acting is magnificent.	he shouldn' t be.
11	screenplay by james l.	his family owns a local diner.	this is not one of them.	random shots of children.	fulltop.	this film is class pure and simple.	yikes!	dir glenn mcquai	i rate this movie a 0.	she looks upward.
12	dallas all over again.	he is finally bond in a video game!	this is one of t.	watch this movie you should enjoy it.	from the sas?	she is simply put, his savior.	yikes!	dialogue editing.	i played that back in junior high.	he wouldn' t, period.
13	wonderfully atmospheric.	he is good in comedy.	this was one of them.	characters start to live in the movie.	plan to return to it at 00.	she plays a prissy little tramp.	yikes!	silence.	i caught this film late at night on hbo.	should be shot.
14	hydrogen filled flying bladders?	g.	this is one of those.	garfield was slimmed down somewhat.	flush now.	this film surprised me a little.	yikes!	silence.	i gave this film a one.	this will be somewhat short.
15	wonderfully acted and superbly cast.	g.	this is one of those.	watch this film with your kids.	plus the dogs are cute too.	this stuff happens all the time.	yakitate!	sidney poitier.	i rate this film 00.	how to tell a story?
16	stanley yelnats iii henry winkle	g.	this is one of those.	giants too, some of us may recall.	formulaic to the max.	this movie starts slow, then tapers off.	yakitate!	kids flying kites.	wait till it appears on tv.	he also paints with love.
17	warming and humble.	he is a good actor.	this is one of those.	and the list of whoppers goes on and on.	also props to this website!	the film sags a bit during the third act.	yikes.	sidesplitting.	i caught it last night on tmc.	he is too talented!
18	dallas leads 0000.	he is good and exudes charm.	this is one of those.	all the stunt work is gorgeous.	plot spoilers!	this movie was quite a pleasant surprise.	yikes.	delusional.	i gave this movie a rating of 0.	her two male foils act well too.
19										

Bibliography

[BB12] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012), pages 281–305.

[Bow+15] S. R. Bowman et al. "Generating Sentences from a Continuous Space". In: *ArXiv e-prints* (November 2015). arXiv: 1511.06349 [cs.LG].

[Bro+16] A. Brock et al. "Neural Photo Editing with Introspective Adversarial Networks". In: *ArXiv e-prints* (September 2016). arXiv: 1609.07093 [cs.LG].

[Chi+17] S. Chiappa et al. "Recurrent Environment Simulators". In: *ArXiv e-prints* (April 2017). arXiv: 1704.02254 [cs.AI].

[Cho+14] K. Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *ArXiv e-prints* (June 2014). arXiv: 1406.1078 [cs.CL].

[Fv14] O. Fabius and J. R. van Amersfoort. "Variational Recurrent Auto-Encoders". In: *ArXiv e-prints* (December 2014). arXiv: 1412.6581 [stat.ML].

[GEB15] L. A. Gatys, A. S. Ecker, and M. Bethge. "A Neural Algorithm of Artistic Style". In: *ArXiv e-prints* (August 2015). arXiv: 1508.06576 [cs.CV].

[Goo+14] Ian Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems 27*. Edited by Z. Ghahramani et al. Curran Associates, Inc., 2014, pages 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[Har28] Ralph VL Hartley. "Transmission of information". In: *Bell Labs Technical Journal* 7.3 (1928), pages 535–563.

[HG14] Clayton J Hutto and Eric Gilbert. "Vader: A parsimonious rule-based model for sentiment analysis of social media text". In: *Eighth international AAAI conference on weblogs and social media*. 2014.

[Hin+12] Geoffrey Hinton et al. "Deep Neural Networks for Acoustic Modeling in Speech Recognition". In: *IEEE Signal Processing Magazine* 29 (November 2012), pages 82–97. URL: https://www.microsoft.com/en-us/research/publication/deep-neural-networks-for-acoustic-modeling-in-speech-recognition/.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Comput.* 9.8 (November 1997), pages 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[JMW14] D. Jimenez Rezende, S. Mohamed, and D. Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *ArXiv e-prints* (January 2014). arXiv: 1401.4082 [stat.ML].

[Kaa+16] C. Kaae Sønderby et al. "Ladder Variational Autoencoders". In: *ArXiv e-prints* (February 2016). arXiv: 1602.02282 [stat.ML].

[Kim+15] Y. Kim et al. "Character-Aware Neural Language Models". In: *ArXiv e-prints* (August 2015). arXiv: 1508.06615 [cs.CL].

[KW13] D. P Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *ArXiv e-prints* (December 2013). arXiv: 1312.6114 [stat.ML].

[LC10] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[LS07] Geoffrey N Leech and Mick Short. *Style in fiction: A linguistic introduction to English fictional prose*. 13. Pearson Education, 2007.

[Maa+11] Andrew L. Maas et al. “Learning Word Vectors for Sentiment Analysis”. In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pages 142–150. URL: http://www.aclweb.org/anthology/P11-1015.

[Mik+13] T. Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *ArXiv e-prints* (October 2013). arXiv: 1310.4546 [cs.CL].

[MP43] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pages 115–133.

[Nyq24] Harry Nyquist. “Certain factors affecting telegraph speed”. In: *Transactions of the American Institute of Electrical Engineers* 43 (1924), pages 412–422.

[PK99] James W Pennebaker and Laura A King. “Linguistic styles: language use as an individual difference.” In: *Journal of personality and social psychology* 77.6 (1999), page 1296.

[RJS17] A. Radford, R. Jozefowicz, and I. Sutskever. “Learning to Generate Reviews and Discovering Sentiment”. In: *ArXiv e-prints* (April 2017). arXiv: 1704.01444 [cs.LG].

[Ros58] Frank Rosenblatt. “The perceptron: A probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65.6 (1958), page 386.

[SHB16] Rico Sennrich, Barry Haddow, and Alexandra Birch. “Controlling politeness in neural machine translation via side constraints”. In: *Proceedings of NAACL-HLT*. 2016, pages 35–40.

[SPG97] Mike Schuster, Kuldip K. Paliwal, and A. General. “Bidirectional recurrent neural networks”. In: *IEEE Transactions on Signal Processing* (1997).

[Sri+14] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pages 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[Wu+16] Y. Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”. In: *ArXiv e-prints* (September 2016). arXiv: 1609.08144 [cs.CL].

[Wu+17] L. Wu et al. “Adversarial Neural Machine Translation”. In: *ArXiv e-prints* (April 2017). arXiv: 1704.06933 [cs.CL].

[WZ89] R. J. Williams and D. Zipser. “A learning algorithm for continually running fully recurrent neural networks”. eng. In: *Neural Computation* 1.2 (1989), pages 270–80, 270–280. ISSN: 1530888x, 08997667. DOI: 10.1162/neco.1989.1.2.270.

[Xu+12] Wei Xu et al. “Paraphrasing for style”. In: *24th International Conference on Computational Linguistics, COLING 2012*. 2012.

[Xu+17] Anbang Xu et al. “A New Chatbot for Customer Service on Social Media”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, Colorado, USA: ACM, 2017, pages 3506–3510. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025496. URL: http://doi.acm.org/10.1145/3025453.3025496.